Turbo Professionnel 5.0

Bibliothèque de routines optimisées pour Turbo Pascal

Volume 1

Ce logiciel et la documentation qui l'accompagne bénéficient de la protection légale accordée aux droits d'auteur. Toute utilisation de ce logiciel suppose que vous (ou votre employeur) ayez conclu un contrat de licence. Pour cela, complétez et renvoyez la licence jointe à ce produit.

Toute reproduction, sous quelque forme que ce soit (y compris électronique ou mécanique, photocopie) est strictement interdite sans l'accord préalable et écrit de INFOLYS-ATEA.

Les informations contenues dans ce document sont sujettes à révision sans préavis.

© Copyright 1987-1988

Logiciel et version anglaise : TurboPower Software

Version française : Infolys-Atea

Marques déposées

TurboPower Software et le logo TurboPower sont des marques de TurboPower Software.

ATEA est une marque déposée, exploitée sous licence par Infolys.

Turbo Professional est une marque déposée de Sunny Hill Software, dont l'utilisation est concédée par licence à TurboPower Software.

Turbo Extender et TurboPower Utilities sont des marques de TurboPower Software.

Turbo Pascal et Turbo Database ToolBox, SideKick et SuperKey sont des marques de Borland International.

Hercules est une marque déposée de Hercules Computer Technology, Inc.

Lotus est une marque déposée de Lotus Development Corporation.

Ms-DoS, OS/2, et Macro Assembler sont des marques de Microsoft Corporation.

IBM PC, XT, AT et PS/2 sont des marques déposées de International Business Machines Corporation.

Table des matières

Volume 1

1. Introduction **1.1**
Configuration matérielle 1.2
Contenu du package 1.2
Mise en route 1.4

2. Gestion d'écrans **2.1**
Généralités 2.1
TPCRT : Unité CRT améliorée 2.4
TPFENETR : Fenêtres "popup" 2.81
TPMENU : Menus déroulants 2.117
TPEDIT : Editeur Ligne 2.168
TPAIDE : Aide adaptée au contexte 2.185
TPSOURIS : Routines de gestion de la souris 2.220
TPPICK : Listes Pick 2.262
TPDIR : Répertoires "popup" 2.282
TPECRAN : Ecrans virtuels 2.289

3. Saisie de données **3.1**
Généralités 3.1
TPSAISIE : Saisie de données 3.3
TPDATE : Variables de type Date et de type Heure 3.96

Index des procédures et fonctions **I.1**

Volume 2

4. Manipulation des chaînes **4.1**
Généralités 4.1
TPCHAINE : Manipulation de chaînes de caractères 4.3
TPASCIIZ : Chaînes longues ASCIIZ 4.62
TPCHNFOR : Chaînes formatées 4.92

5. Appels au DOS et au BIOS **5.1**
Généralités 5.1
TPDOS : Appels au DOS ou au BIOS 5.3
TPCLPLUS : Interface de clavier étendu 5.35

6. Interruptions et programmes résidents **6.1**
Généralités 6.1
TPINT : Routines de gestion d'interruption (ISR) 6.4
TPTSR : Gestion des programmes résidents 6.21
TP8087 ..... 6.51
TPINT24 : Gestionnaire d'erreurs critiques 6.58

7. Tri **7.1**
Généralités 7.1
TPTRI : Routine de tri en mémoire 7.4

8. Allocation mémoire **8.1**
Généralités 8.1
TPEMS : Gestion de la mémoire EMS 8.4
TPMEMEXT : Gestion de la mémoire étendue 8.20

9. Macros **9.1**
Généralités 9.1
TPMACRO : Macros clavier 9.3
TPEDMAC : Editeur de macros 9.20

10. Grands tableaux **10.1**
Généralités 10.1
TPTBMRAM : Tableaux en RAM normale 10.4
TPTBMEXP : Tableaux en mémoire EMS 10.20
TPTBMVIR : Tableaux en mémoire virtuelle 10.33

11. Arithmétique BCD **11.1**
Généralités 11.1
TPBCD : Réels en haute précision 11.3

12. Divers **12.1**
Généralités 12.1
TPEREXEC : Gestion des erreurs d'exécution 12.2
TPINLINE : Macros inline 12.7

13. Programmes de démonstration **13.1**
SMACS : Processeur de macros 13.2
CALC : Calculatrice du programmeur 13.6
GESTEMPS : Utilitaire de gestion d'emploi du temps 13.11
GENETAT : Générateur d'états pour GESTEMPS 13.14
REF : Table synoptique des caractères 13.17
CREEMENU : Générateur de menus 13.19
BTRI : Utilitaire de tri de fichier 13.30
CHERCHE : Utilitaire de recherche dans des fichiers textes 13.31
REPETE : Répétiteur de commandes 13.33
DIFF : Recherche des différences 13.36
VBROWSE : "Browse" de fichiers virtuels 13.37
SAISIE : Exemple de saisie de données 13.39
Autres programmes 13.44

14. Annexes **14.1**
Ordre d'écriture des unités 14.1
Les nouveautés de Turbo Professionnel 5.0 ..... 14.4
Gestion du clavier avec Turbo Professionnel 14.6
Utilisation des "overlays" avec Turbo Professionnel 14.16

Index des procédures et fonctions **I.1**

1 Introduction

Notre premier regard sur *Turbo Professionnel* 4.0 nous a conduit à deux réflexions. D'une part, *Borland* a considérablement développé le marché potentiel des bibliothèques de programmes grâce au concept de la compilation séparée. D'autre part, les nombreuses différences entre *Turbo Pascal* 4.0 et *Turbo Pascal* 3.0 ont gêné les programmeurs qui ont passé trop de temps à développer et à optimiser leurs routines avec le compilateur précédent. Ce que nous avions de mieux à faire, il nous semble, était de développer une puissante bibliothèque de routines qui permettrait aux programmeurs *Turbo Pascal* avertis de convertir facilement et rapidement leurs programmes.

Ainsi naquit *Turbo Professionnel* 4.0, la bibliothèque de programmes la plus importante. Elle remporta un succès immédiat en Amérique et dans le monde entier. Nous avons reçu une centaine d'appels et de lettres de clients satisfaits vantant simplement les qualités du produit et nous les en remercions. La plupart de ces courriers réclamaient à l'unisson de nouvelles fonctionnalités : écran de saisie de données, gestion de la souris, aide, fenêtres qui explosent, etc... Certaines demandes ont été immédiatement satisfaites, mais dans de nombreux cas, cela n'a pas été possible. Merci et comptez sur nous pour vous donner satisfaction dans la prochaine version.

En découvrant *Turbo Pascal* 5.0, nous avons pensé qu'il serait préférable d'améliorer *Turbo Professionnel* 4.0 plutôt que de créer une nouvelle bibliothèque de programmes. *Turbo Professionnel* 5.0 était né : les chiffres défient notre imagination : ***600*** routines (contre 400), ***31*** unités (contre 25), ***750*** pages de documentation répartis en deux volumes (contre 450 pages en un seul volume). Les principales innovations correspondent aux demandes de nos clients. Vous y retrouverez certains de vos propres besoins.

Afin de bien débuter avec *Turbo Professionnel* 5.0, nous vous conseillons selon votre expérience du produit de procéder comme suit :

- Si vous êtes un utilisateur averti, lisez l'appendice B "les nouveautés de *Turbo Professionnel* 5.0" qui présente brièvement les nouvelles caractéristiques. Passez ensuite à la section "démarrage" du chapitre 1 qui décrit le mode d'installation de *Turbo Professionnel* 5.0.

- Si vous êtes un nouvel utilisateur, passez directement à la section "démarrage" du chapitre 1, qui vous fournira également les directives générales d'utilisation de la bibliothèque.

Avant de commencer, veuillez consulter la table des matières qui vous donnera une idée de la structure de cette documentation.

Excepté l'introduction (chapitre 1), le volume 1 traite presque exclusivement les routines de gestion d'écrans (chapitre 2 et 3). En fin de volume, vous trouverez deux index : un index général par mots clés et un index de toutes les procédures et fonctions de la bibliothèque. Le volume 2 reproduit la table des matières et contient l'ensemble des chapitres couvrant les autres unités de la bibliothèque (chapitre 4-13). Le chapitre 14 contient des appendices, des explications détaillées de thèmes d'intérêt général. Le volume se termine par une reproduction des deux index du volume 1.

Veuillez consulter la licence d'utilisation au chapitre 1. Elle précise vos droits de propriété sur *Turbo Professionnel* 5.0 ainsi que la nature et la portée de la garantie.

Configuration matérielle

Pour utiliser *Turbo Professionnel*, vous devez posséder :

1. Un IBM PC, XT, AT (ou compatible) ou un PS/2 fonctionnant sous DOS 2.x ou 3.x.
2. *Turbo Pascal* version 4.0 ou 5.0
3. 256 K de mémoire (Notez cependant que le compilateur *Turbo Pascal* peut demander plus de mémoire).
4. 2 lecteurs de disquettes ; un disque dur est cependant conseillé.
5. Un moniteur 80 colonnes.

En option :

6. Le Turbo Assembleur version 1.0 ou supérieure de Borland (TASM) ou le Macro Assembleur Microsoft (MASM) 4.0 ou plus. Ils sont obligatoires si vous désirez modifier un programme assembleur.

Contenu du package

Turbo Professionnel est livré sur quatre disquettes. La première contient des programmes de démonstration qui utilisent la bibliothèque de *Turbo Professionnel* ; la seconde contient le source Pascal de la plupart des unités ;

la troisième contient le source assembleur des librairies et le code source des autres unités ; la quatrième contient le code source de tous les programmes de démonstration, ainsi qu'un répertoire BONUS contenant des unités et des programmes du domaine public. Les fichiers ne sont pas protégés et nous vous recommandons de faire une copie des disquettes avant de les utiliser.

Les programmes de démonstration et la bibliothèque *Turbo Professionnel* ont été compressés avec l'utilitaire du domaine public ARC. Nous fournissons le programme ARCX avec lequel vous pourrez désarchiver les fichiers.

Vous trouverez les fichiers suivants sur les disquettes d'origine.

Disquette DEMO

ALIRE	contient les dernières mises à jour de la documentation.
CLFIX.COM	petit utilitaire permettant aux programmes qui utilisent l'unité TPCLPLUS de détecter la présence d'un clavier étendu lorsque les tests standards sont défaillants.
SAISIE.EXE	programme de démonstation de la saisie en mode format et des aides "popup".
SAISIE.AID	fichier d'aide pour SAISIE.EXE.
INSTALL.EXE	programme d'installation pour *Turbo Professionnel* 5.0.
CREEMENU.EXE	utilitaire d'aide à la création des menus avec l'unité TPMENU.
REF.EXE	programme résident affichant les codes ASCII, les touches, etc...
SMACS.EXE	processeur de macros semblable à SuperKey de Borland.
TPRO.MAK	fichier à utiliser avec avec l'utilitaire MAKE de Borland (fourni avec *Turbo Pascal*). Il fournit les directives permettant à MAKE de restructurer les unités et les programmes de démonstration de *Turbo Professionnel*.
TURBO5.MAC	exemple de macros SMACS à utiliser avec l'éditeur de *Turbo Pascal*.
DEROULFE.EXE	démonstration de fenêtres "popup".

Disquette SRC1

ARCX.COM	version réduite de l'utilitaire ARC qui contient seulement la fonction de désarchivage. Tapez **ARCX** puis (ENTER) pour obtenir le menu du programme.
TPSRC1.ARC	code source Pascal archivé de la plupart des routines de la bibliothèque.

Disquette SRC2

ARCX.COM	seconde copie de ARCX.
TPASM.ARC	code source archivé des routines écrites en assembleur.
TPOBJ.ARC	code objet archivé de tous les programmes assembleur.
TPSRC2.ARC	code source Pascal archivé des autres routines de la bibliothèque.

Disquette SRC3

ARCX.COM	troisième copie de ARCX.
DEMOSRC.ARC	code source Pascal archivé de tous les programmes de démontration.
BONUS	répertoire contenant de nombreux programmes et unités du dommaine public. Plusieurs ont été écrits par *TurboPower Software*, les autres ont été ajoutés avec l'autorisation des auteurs. Consultez le fichier ALIRE pour plus de détails.

Mise en route

Avant de poursuivre, nous allons énumérer rapidement les notions de bases nécessaires pour lire cette documentation. Nous supposons non seulement que vous savez mettre en oeuvre le compilateur *Turbo Pascal* (4.0 ou 5.0) mais également et surtout que vous êtes familiarisé avec la version 4.0 (ou les ajouts de la version 5.0), notamment :

- Le concept d'unités (fichiers TPU)
- La signification de l'instruction USES ("USES CRT, DOS;")
- Les différences entre les appels NEAR et FAR ({$F+}vs.{$F-})
- Les différences entre les parties interface et implémentation d'une unité
- L'utilisation du symbole "@" ("MonPointeur := @MaVariable;")
- Les différences entre les notions de code, de données, de pile et de tas.

Vous pourriez trouver certaines parties de cette documentation plus complexes qu'elles ne le sont en réalité si vous ne possédez pas les connaissances de base sur les unités, les segments mémoire et les pointeurs. Veuillez consulter le manuel *Turbo Pascal* selon votre niveau.

De plus, nous supposons acquises les connaissances sur :

- les procédures EXIT ("ExitProc := @MonExitProc;")
- les blocs d'initialisation d'une unité (Bloc "begin..end" à la fin d'une unité)
- les procédures d'interruptions
- les procédures INLINE appelées *macros inline* (vous n'aurez pas à les écrire vous-même mais seulement à les comprendre)
- les notions d'interruption et de vecteur d'interruption
- les notions de programmes résidents en mémoire (TSR "TERMINATE AND STAY RESIDENT")
- les directives de compilation conditionnelle ({$IFDEF}, {$ENDIF}, etc).

Néanmoins, *Turbo Professionnel* n'est pas destiné uniquement aux programmeurs expérimentés ; il est conçu pour être utilisé par des débutants en *Turbo Pascal*. Les modes de fonctionnement particuliers vous seront expliqués dans certains cas.

La bibliothèque *Turbo Professionnel* est composée d'environ ***600*** routines réunies dans ***31*** unités :

Gestion d'écran

L'unité standard CRT est remplacée par l'unité TPCRT qui contient de nouvelles routines. TPECRAN traite des écrans virtuels et des zones de mémoire dynamique assimilées à des écrans. TPFENETR contient des routines de gestion des fenêtres et des écrans virtuels. TPMENU vous permet de créer rapidement et facilement des menus élaborés, notamment avec l'utilitaire CREEMENU. TPEDIT fournit un éditeur plein écran. TPAIDE vous permet de créer des systèmes d'aide contextuelle facilement. TPSOURIS simplifie la gestion de la souris dans vos programmes. TPPICK affiche des "listes historiques", c'est-à-dire des fenêtres "popup" contenant une liste d'éléments à sélectionner. TPDIR conjuguée avec TPPICK permet de créer des listes "popup" de noms de fichiers. TPECRAN offre une autre méthode de traitement des écrans virtuels (cette unité sert principalement à garantir la compatibilité avec les versions antérieures de *Turbo Professionnel*).

Saisie de données

TPSAISIE offre des facilités de saisie en mode format. TPDATE contient des routines de traitement des dates et de l'heure.

Gestion des chaînes de caractères

TPCHAINE contient un ensemble de routines de base pour manipuler les chaînes de caractères standard de Turbo. TPASCIIZ vous permet aussi de manipuler aisément des chaînes de longueur supérieure à 255 caractères. TPCHNFOR vous permet d'utiliser les instructions d'écriture standard pour créer des chaînes formatées.

Interfaces DOS/BIOS

TPDOS vous permet d'accéder aux services du DOS et du BIOS non accessibles par l'unité standard DOS. TPCLPLUS facilite l'écriture des programmes qui utilisent un clavier étendu.

Interruptions et programmes résidents

TPINT gère les routines d'interruption à votre place. TPTSR contient tout ce dont vous avez besoin pour écrire, sans trop d'efforts, des programmes résidents de qualité. TP8087 contient les routines de bas niveau nécessaires lorsque vous utilisez des programmes résidents qui utilisent le processeur 8087 ou l'émulation virgule flottante de *Turbo Pascal* 5.0, ainsi que les routines qui vous permettent de traiter les erreurs fatales sur des claculs arithmétiques. TPINT24 fournit un gestionnaire d'erreurs du DOS qui évitera à vos utilisateurs d'utiliser les options Abandonne, Recommence ou Ignore.

Tri

TPTRI comporte un programme de tri rapide en mémoire qui fonctionne avec n'importe quel type de données.

Allocation mémoire

TPEMS vous permet d'accéder facilement à la mémoire étendue tandis que TPMEMEXT autorise le mode protégé sur les machines 286 et 386 et vous permet d'utiliser la mémoire supplémentaire pour le stockage des données.

Macros

TPMACRO permet d'ajouter facilement des macros à vos programmes et TPEDMAC contient un éditeur de macros élaboré.

Grands tableaux

Trois unités brisent la frontière des 64 K pour les tableaux. TPTBMRAM utilise la mémoire RAM normale pour les accès rapides. TPTBMEXP utilise la mémoire étendue pour autoriser les tableaux d'au plus 8 mégaoctets et TPTBMVIR enregistre les éléments de tableaux sur disque ("mémoire virtuelle") pour autoriser des tableaux jusqu'à 32 mégaoctets.

Arithmétique BCD

TPBCD fournit un ensemble de routines pour des calculs de haute précision en utilisant les réels BCD (Binary Coded Decimal).

Routines diverses

TPEREXEC vous permet de récupérer la plupart des erreurs d'exécution. TPINLINE.INC contient une palette de *macros inline* utiles pour travailler avec des pointeurs, indicateurs binaires et tables de sauts (et beaucoup d'autres choses).

Commençons par apprendre à transférer les fichiers et les disquettes distribuées avec *Turbo Professionnel* sur votre propre disque.

Si vous possédez déjà *Turbo Professionnel* 4.0 et que vous désirez installer la version 5.0, nous vous recommandons, soit d'effacer tous les fichiers de la version 4.0, soit de créer un autre répertoire pour les nouveaux fichiers. Les fichiers de la version 5.0 annulent et remplacent tous ceux de la version 4.0. Si vous avez modifié les unités de votre version 4.0, n'oubliez pas d'en faire une copie pour reporter ces modifications dans la version 5.0 si nécessaire.

Vous ne trouverez pas de fichiers TPU avec cette version. En effet, les formats des fichiers TPU *de Turbo Pascal* 4.0 et *Turbo Pascal* 5.0 ne sont pas compatibles et *Turbo Professionnel* doit pouvoir fonctionner avec les deux générations du compilateur.

Aussi, vous devez compiler les unités de *Turbo Professionnel* et nous vous présentons deux méthodes :

- La première, la plus facile à mettre en oeuvre, utilise le programme INSTALL.EXE fourni sur la disquette DEMO. Il permet de copier automatiquement n'importe quel nombre d'unités, de les désarchiver et de les compiler où vous voulez sur votre disque.

  Le programme INSTALL est particulièrement utile si vous ne possédez pas de disque dur, si vous ne souhaitez pas installer toutes les unités ou si vous n'êtes pas familiarisé avec les fichiers "MAKE" style UNIX.

- La seconde méthode, plus puissante mais moins conviviale, s'appuie sur l'utilitaire Borland MAKE.EXE et le fichier "MAKE" TPRO.MAK.

Le fichier MAKE contient tous les enchaînements nécessaires pour les unités *Turbo Professionnel*. Cette méthode est recommandée si vous désirez installer tous les outils *Turbo Professionnel* sur un disque dur disposant d'au moins 3 Méga-octets. Si besoin, il compile et réassemble les fichiers.

Ces deux méthodes impliquent que vous possédiez au moins deux lecteurs de disquettes ou un lecteur et un disque dur. Si vous ne possédez qu'un seul lecteur de disquettes sans RAM disque, vous devrez créer votre propre méthode d'installation.

Méthode d'installation avec fichier MAKE

1. Tout d'abord, sélectionnez ou créez un sous-répertoire pour les fichiers *Turbo Professionnel*. Il contiendra les codes sources Pascal et Assembleur, les fichiers OBJ et TPU, les fichiers EXE des programmes de démonstration et des utilitaires. Vous pouvez spécifier le chemin d'accès à ce répertoire aussi bien pour le compilateur *Turbo Pascal* que pour le chemin d'accès DOS (par exemple pour activer directement des utilitaires comme SMACS et CREEMENU).

   Une fois *Turbo Professionnel* installé, vous pouvez aussi transférer les fichiers EXE dans un autre répertoire déjà connu dans le chemin d'accès DOS.

   N'oubliez pas que vous devez disposer d'au moins 3 méga-octets sur votre disque pour mettre en oeuvre cette méthode d'installation.

   Note : Une fois l'installation terminée, vous pouvez effacer de nombreux fichiers ce qui ramènera l'espace disque à moins de 500 Ko, si vous ne conservez que les fichiers TPU. Si vous disposez de moins d'espace disque, utilisez l'autre méthode.

2. Sélectionnez le répertoire destiné à *Turbo Professionnel* 5.0 et recopiez tous les fichiers du répertoire racine de chaque disquette *Turbo Professionnel* en procédant comme suit.

   Placez la disquette DEMO dans le lecteur A :

   ```
   Copy A: *.*
   ```

   Placez la disquette SRC1 dans le lecteur A :

   ```
   Copy A: *.*
   ```

   Placez la disquette SRC2 dans le lecteur A :

   ```
   Copy A: *.*
   ```

   Placez la disquette SRC3 dans le lecteur A :

   ```
   Copy A: *.*
   ```

3. Désarchivez les fichiers compressés en tapant :

```
ARCX *.ARC
```

Si un nom de fichier à désarchiver existe déjà dans votre répertoire, le programme ARCX vous demandera de confirmer pour poursuivre. Pressez (CTRL)-C pour arrêter et revenir à l'invite du DOS.

4. Supprimez si vous le souhaitez, les fichiers ARC pour gagner de la place :

```
DEL *.ARC
```

Avant de poursuivre, adaptez TPRO.MAK à votre système pour tenir compte éventuellement du chemin d'accès au compilateur en ligne de *Turbo Pascal* et de votre assembleur. Le fichier contient toutes les instructions, utilisez simplement votre éditeur favori.

Si vous hésitez à modifier même légèrement un fichier MAKE, utilisez l'autre méthode d'installation.

5. Vous pouvez maintenant compiler ce que vous voulez avec l'utilitaire MAKE de Borland :

```
MAKE-FTPRO.MAK
```

N'essayez pas d'utiliser le MAKE de Microsoft (ou tout autre utilitaire de ce type) avec le fichier MAKE qui vous est fourni. Les différences provoqueraient des erreurs de fonctionnement.

Le fichier TPRO.MAK ainsi adapté, MAKE créera toute la bibliothèque correctement. En cas d'erreur, recontrôlez le fichier MAKE. Si cela n'aboutit pas, vérifiez s'il existe un fichier TPC.CFG (vous trouverez la description de ce fichier dans le manuel *Turbo Pascal*). Si un tel fichier existe, effacez-le ou renommez-le temporairement pendant l'exécution de MAKE.

Si vous n'avez pas assez de place sur votre disque, vous pouvez maintenant effacer de nombreux fichiers (tant que vous n'avez pas besoin de modifier ou de restructurer une unité), éventuellement tapez :

```
DEL TP*.PAS
DEL TP*.ASM
DEL TP*.IN
DEL TP*.OBJ
DEL BCD*.*
```

Vous ne conserverez ainsi que les fichiers *Turbo Professionnel* 4.0 importants et les fichiers sources et EXE des programmes de démonstration.

Si cela n'a pas été encore fait, vous pouvez encore imprimer et lire le fichier ALIRE qui fournit les mises à jour de cette documentation.

Méthode d'installation avec INSTALL.EXE

Nous avons écrit ce programme pour vous faciliter le désarchivage, la recopie et la compilation des unités *Turbo Professionnel.* Vous pouvez sélectionner seulement les fichiers dont vous avez besoin. INSTALL gère même les dépendances entre unités à votre place, si bien que vous disposez de tous les fichiers nécessaires pour compiler un programme donné.

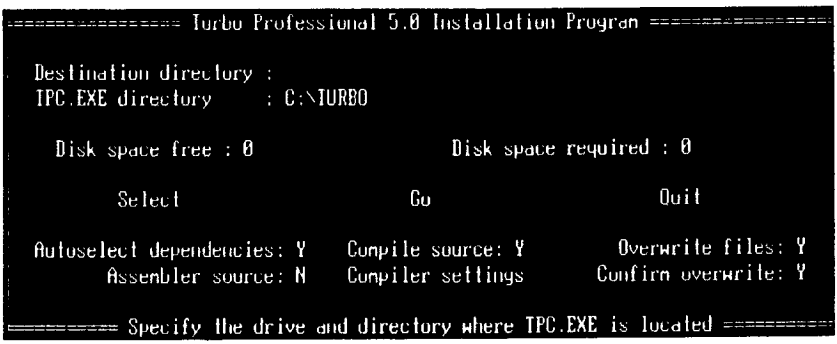
1. Créez un sous-répertoire sur votre disque dur ou formatez quelques disquettes. Le programme INSTALL vous indiquera l'espace nécessaire en octets pour les fichiers que vous sélectionnez afin que vous puissiez prendre vos dispositions pour poursuivre si à un moment donné l'espace disque devenait insuffisant.

2. Placez la disquette DEMO dans le lecteur A et tapez :

   ```
   A: INSTALL
   ```

   Au démarrage, INSTALL mémorise le lecteur et le répertoire courant qu'il utilisera comme destination par défaut (si le lecteur initial est A, la destination prise en compte sera C:\TPRO5). INSTALL recherche également le chemin d'accès DOS à votre copie de TPC.EXE qu'il utilisera pour compiler les unités *Turbo Professionnel.*

3. Vous devrez voir apparaître ce qui suit sur votre écran :



   Les deux premières lignes affichent le répertoire par défaut, si vous voulez modifier ces informations ou si INSTALL ne trouve pas TPC.EXE, placez la barre lumineuse avec les touches curseur sur le répertoire à modifier. Pressez (ENTER) pour le sélectionner, tapez le nouveau répertoire et pressez (ENTER) à nouveau pour valider l'opération. Pour interrompre une modification, pressez (ESC).

Tapez seulement un nom de lecteur et de répertoire et non un chemin d'accès complet. Vérifiez bien que le répertoire spécifié pour TPC.EXE contient la bonne version du compilateur si vous possédez les deux versions 4.0 et 5.0.

Avec un système à deux lecteurs de disquettes, vous taperez probablement B:\ comme répertoire destination et A:\ comme répertoire destiné à TPC.EXE. Dans ce cas, vous échangerez les disquettes d'origine de *Turbo Professionnel* dans le lecteur A et placerez les disquettes formatées dans le lecteur B.

Au moment de compiler, vous devrez placer dans le lecteur A une disquette contenant TPC.EXE et TURBO.TPL. Si vous disposez d'un espace RAM limité, vous aurez besoin d'une disquette contenant COMMAND.COM au cas où le DOS ait besoin de recharger ce fichier après compilation.

La ligne suivante affiche le nombre d'octets disponibles sur le lecteur destination qui sera actualisé lors des recopies et compilations de fichiers sur ce lecteur. Sur la même ligne, est affiché le nombre d'octets nécessaires pour écrire les fichiers *Turbo Professionnel* sélectionnés sur le lecteur de destination. Vérifiez que le nombre d'octets disponibles est supérieur au nombre d'octets nécessaires. INSTALL vous laisse libre de sélectionner un ensemble de fichiers qui remplissent ces conditions.

La quatrième ligne de couleur différente affiche les boutons de mise en oeuvre. Lorsque vous placez la barre lumineuse sur l'un de ces boutons et pressez (ENTER), INSTALL enchaîne sur d'autres actions.

Bouton SELECT

INSTALL affiche une deuxième fenêtre, la fenêtre de sélection de fichier. Elle vous permet de choisir les programmes de démonstration et les unités Turbo Professionnel que vous désirez installer. Déplacez la barre lumineuse avec les flèches, pressez (INS) pour sélectionner une unité donnée, la touche (DEL) pour désactiver une unité, la touche (ENTER) comme bascule de sélection ou (CTRL)-(ENTER) pour annuler les sélections en cours. Les unités sélectionnées sont affichées avec une flèche à gauche. Observez que le compteur d'espace disque est mis à jour après chaque sélection. Lorsque vous sélectionnez une unité ou un programme de démonstration particulier, INSTALL prend réellement en compte l'ensemble des fichiers nécessaires pour compiler cette unité. (fichier Pascal, fichier inclus, fichier OBJ, etc). Si l'option d'auto-sélection de dépendance est activée, INSTALL sélectionne toutes les unités dont dépend le programme ou l'unité en cours. Vous regarderez avec amusement le mouvement des unités appelées dans chaque programme. Faites-le en sélectionnant et en pressant alternativement (CTRL)-(ENTER).

Le mécanisme de sélection automatique de dépendance ne fonctionne que dans un sens. Lorsque vous désactivez une unité, les unités dont elle dépend ne sont pas désactivées en même temps.

Lors de l'écriture sur disque tous les résultats relatifs à chaque ensemble de sélection de fichiers, doivent tenir sur une disquette.

C'est le moment de rappeler qu'il est très important de recopier le fichier ALIRE. Ce fichier contient des instructions qui ne figurent pas dans la documentation, nous ne comptons plus le nombre d'appels de clients dont les réponses à leur problème se trouvaient dans le fichier ALIRE !

Pressez (CTRL) pour revenir à la fenêtre de commande lorsque vous avez sélectionné vos fichiers.

Bouton GO : exécution du désarchivage, de la recopie et de la compilation

Bouton QUITTE : retour à l'invite du DOS.

Avant d'utiliser le bouton GO lisez ce qui suit.

Les deux dernières lignes de la fenêtre de commande affichent les bascules de contrôle du fonctionnement du programme INSTALL. Placez la barre lumineuse sur l'une des options et pressez (ENTER) pour la basculer.

L'option d'auto-sélection de dépendance a déjà été expliquée.

Lorsque l'option "Assemble les sources" a pour valeur X, INSTALL désarchive et recopie les fichiers sources en assembleur dans leur destination, sinon seul les fichiers OBJ sont recopiés. Vous n'avez pas besoin du code source assembleur sauf si vous voulez l'étudier ou le modifier.

Lorsque l'option 8087 a pour valeur X, INSTALL compile toutes les unités des programmes avec la directive $N+. Le coprocesseur mathématique (ou l'émulateur logiciel 8087) est activé. La plupart des unités ne sont pas concernées par cette option mais certaines comme TPCHAINE, TPEDIT, TPSAISIE, et TP8087 le sont .

Lorsque la sélection OVERWRITE a la valeur Y, INSTALL écrase tout fichier de même nom dans le répertoire de destination si l'option "Confirme" a également pour valeur X. En positionnant OVERWRITE à N, vous pouvez facilement ajouter d'autres fichiers dans votre répertoire Turbo Professionnel sans perte de temps avec les fichiers qui existent déjà.

Lorsque vous avez renseigné toutes les options et sélectionné les fichiers désirés, vous pouvez activer le bouton GO. INSTALL vous demandera d'entrer les disques *Turbo Professionnel* concernés. Vous pourrez suivre le déroulement de l'installation dans la fenêtre DOS au bas de l'écran. En cas de problème, INSTALL vous demandera si vous voulez poursuivre ou arrêter les initialisations en cours. Lorsque l'installation se termine bien, pressez (ESC) pour effacer le message de statut puis sélectionnez le bouton QUITTE pour revenir au DOS ou sélectionnez un autre ensemble de fichiers.

Vous venez d'apprendre, dans l'ensemble, à compiler un programme qui utilise la bibliothèque *Turbo Professionnel.*

Maintenant, voyons comment écrire un programme qui utilise la bibliothèque. C'est très simple en vérité. Vous devez simplement déclarer dans une instruction USES toutes les unités relatives aux routines que vous voulez appeler dans votre programme.

```
program DemoEcranVirtuel;
uses
   TpCrt,
   TpFenetr;
var
   EV  : EcranV;
   Ligne,
   Col : Byte;
   Touche : Word;
   S   : String;
begin
   {Créé un écran virtuel de 100 LigneH et 255 colonnes}
   if not CreeEcranV(EV, 100, 255) then Halt(1);
   {Remplit l'écran virtuel}
   S[0] := #255;
   for Col := 1 to 255 do S[Col] := Char(Col);
   ActiveEcranV(EV);
   for Ligne := 1 to 100 do EcritVite(S, Ligne, 1, FixeCouleur(Ligne
            and $F));
   DesactiveEcranV;
   {Créé une fenêtre et affiche quelques instructions}
   CursCache;
   Window(1, 1, LargeurEcran, HauteurEcran - 1);
   EcritVite('       Utilisez les touches de déplacement du curseur ou
            ESC pour quitter       ', HauteurEcran, 1, $E);
   {Copie une partie de l'écran virtuel sur l'écran physique}
   DeplaceEcrVDansFen(EV, 1, 1);
   {Déplacement du curseur jusqu'à ce que ESC soit pressé}
   Ligne := 1; Col := 1;
```

```
    repeat
       Touche := LitMotTouche;
       case Touche of
          $4800 : {Haut}
          if Ligne > 1 then begin
             Dec(Ligne);
             DeplaceEcrVDansFen(EV, Ligne, Col);
          end;
          $4B00 : {Gauche}
          if Col > 1 then begin
             Dec(Col);
             DeplaceEcrVDansFen(EV, Ligne, Col);
          end;
          $4D00 : {Droite}
          if Col < 255 - Pred(LargeurEcran) then begin
             Inc(Col);
             DeplaceEcrVDansFen(EV, Ligne, Col);
          end;
          $4500 : {Bas}
          if Ligne < 100 - (HauteurEcran - 2) then begin
             Inc(Ligne);
             DeplaceEcrVDansFen(EV, Ligne, Col);
          end;
       end;
    until Lo(Touche) = 27;
    {restitution de l'écran}
    CursNormal;
    Window(1, 1, LargeurEcran, HauteurEcran);
    ClrScr;
end.
```

Saisissez et exécutez ce programme pour illustrer les capacités de TPFENETR pour gérer des écrans virtuels supérieurs à l'écran physique.

Lorsque vous écrirez vos programmes, vous aurez à déterminer les routines et les unités nécessaires. A la longue, vous saurez vous débrouiller tout seul, mais pour l'instant, nous allons vous aider.

Vous disposez pour votre recherche de deux éléments. D'une part, les références et explications relatives à chaque routine (chapitres 2 et 12), classée alphabétiquement par section. Les noms des routines (au plus 2 par page) apparaissent en caractères gras en haut de chaque page. D'autre part, vous disposez d'un index des procédures et fonctions à la fin de chaque volume qui est très utile lorsque vous connaissez le nom de la routine qui vous intéresse.

Supposons par exemple, que vous ayez besoin de revoir les spécifications de la routine *CreeEcranV*. Si vous avez oublié son nom, parcourez l'unité TPFENETR jusqu'à ce que vous la trouviez. Grâce à la qualité de la typographie, cela est très rapide. Sinon, consultez l'index qui vous renverra au numéro de page dans l'unité concernée. Vous y trouverez outre la description de la routine et son mode d'emploi, les noms des routines relativement proches telle que *ActiveEcranV*.

Si vous n'êtes pas convaincu des capacités de *Turbo Professionnel* 5.0, vous pouvez tout d'abord lire les généralités des chapitres 2 et 12 ; vous y trouverez des explications plus complètes sur chaque unité et sur leur mode d'emploi. Ensuite, essayez de compiler et d'exécuter certains programmes de démonstration (certains sont disponibles en version précompilée sur la disquette de démonstration). Consultez la documentation relative à ces programmes au chapitre 13.

2 Gestion d'écrans

Généralités

TPCRT remplace complètement l'unité standard CRT de *Turbo Pascal*. Bien que totalement compatible avec CRT, TPCRT offre un nombre impressionnant de nouveautés et d'améliorations. Vous y trouverez toutes les routines pour déplacer le curseur et le changer de forme ; pour lire des caractères et des chaînes de caractères à partir de l'écran ; pour écrire directement et rapidement dans la mémoire vidéo ; pour sauvegarder et enregistrer des fenêtres ; pour sauvegarder des écrans compactés sur disque ; pour passer du mode EGA à 43 lignes au mode VGA à 50 lignes sans effacer l'écran ; des routines de gestion du clavier. TPCRT vous permet également d'écrire sur n'importe quelle page vidéo avec un adaptateur couleur (même avec *WriteLn*), et supporte des systèmes multi-tâches tels que *DesqView* et *OmniView*.

L'unité TPFENETR contient un gestionnaire de fenêtres "popup" complet, ainsi que les routines dont vous aurez besoin pour implémenter des écrans virtuels (TPECRAN utilise une approche différente). Le gestionnaire de fenêtres permet d'empiler facilement des fenêtres. De simples fonctions d'appels vous permettent de définir une fenêtre ; d'effacer une fenêtre, de réafficher une fenêtre, de détruire une fenêtre, de déplacer une fenêtre du bas de la pile au sommet de la pile, de déplacer ou de recalculer la taille d'une fenêtre, etc... Les fenêtres peuvent exploser sur l'écran (avec des effets sonores) et être entourées de zones ombrées. Essayez d'exécuter DEROULFE.EXE pour voir une démonstration éclatante des possibilités de TPFENETR.

TPMENU vous aide à créer des menus déroulants facilement. TPMENU offre de nombreuses options : menus déroulants de type Borland ou menus classiques (barre horizontale) de type Lotus ; des menus qui se déroulent horizontalement ou verticalement ; des menus personnalisés qui affichent les informations sur l'état de votre programme avec au plus six couleurs

différentes par menu, un ensemble d'options pour les types de cadres, des menus ombrés et des menus qui explosent. Toutes ces options peuvent néanmoins vous paraître confuses. Mais si vous trouvez les routines de TPMENU difficiles à comprendre au premier abord, ne vous découragez pas.

TPEDIT contient un éditeur réduit mais entièrement compatible *WordStar* que vous pouvez facilement inclure dans vos programmes. La routine *LitChaine* implémente les déplacements de base du curseur et les commandes orientées ligne, caractère et mot disponibles dans un éditeur de texte plein écran : déplacement du curseur à gauche et à droite, au début ou à la fin de ligne, à gauche et à droite d'un mot, effacement de caractères, effacement de mots, effacement de la fin de la ligne, effacement de toute la ligne, restauration de ligne, mode insertion et recouvrement. *LitChaine* permet aussi de dérouler les champs à modifier horizontalement lorsqu'ils sont plus longs que l'écran. Des versions spéciales de la routine *LitChaine* sont également incluses pour la sortie des chaînes numériques (*LongInt*, *Integer*, *Word* et *Real*), ainsi que deux routines semblables que vous pouvez utiliser pour saisir des caractères individuels.

TPSOURIS offre un ensemble de routines de bas niveau qui simplifie la gestion de la souris dans vos programmes. Vous trouverez de nombreuses routines pour lire les boutons de la souris, calculer la position du pointeur de la souris, le cacher et le faire réapparaître etc... Bien mieux, de nombreuses autres unités - TPEDIT, TPCRT, TPAIDE, TPMENU et TPPICK - utilisent les routines de TPSOURIS pour gérer, à votre place, l'interface de la souris.

TPAIDE contient des routines qui vous permettent de développer des systèmes d'aide "POPUP" sophistiqués rapidement et facilement. Entre autre finesse, TPAIDE comporte un système d'indexation automatique des thèmes facilitant l'imbrication et la liaison de ces thèmes à travers les textes d'aide, un support intégré de la souris et une routine qui recherche dans l'écran, les mots clés correspondant à un point d'entrée dans l'index.

L'unité TPPICK gère les "listes Pick" que vous pouvez consider comme des "menus dynamiques".

TPPICK permet d'afficher des listes d'éléments interdépendants et de sélectionner l'élément de votre choix.

L'unité conjointe, TPDIR, affiche une liste ordonnée des noms de fichiers vérifiant un critère donné et permet à l'utilisateur d'en choisir un. Si vous êtes familiarisé avec les environnements intégrés des produits BORLAND et que vous souhaitez inclure de telles facilités dans vos programmes, TPDIR vous conviendra parfaitement.

L'unité TPECRAN se situe au-dessus de TPCRT et possède un nombre de caractéristiques supplémentaires, notamment des *écrans virtuels*. Ceci signifie que vous pouvez définir une portion de la mémoire comme un écran, y écrire puis le placer sur l'écran actuel lorsque vous avez terminé. TPECRAN possède aussi des routines pour déplacer des fenêtres horizontalement et verticalement, recopier un écran vers un autre, remplir une fenêtre avec un caractère particulier à une vitesse ultra-rapide. Vous pourrez préférer cette unité à TPFENETR car elle offre quelques possibilités supplémentaires avec les écrans virtuels.

TPCRT : unité CRT améliorée

Ainsi que nous l'avons vu au début du chapitre, TPCRT est une version améliorée de l'unité CRT de Turbo. Si vous n'utilisez aucune de ces extensions, vos programmes fonctionneront normalement comme si vous utilisiez simplement CRT. Elle comporte néanmoins des différences importantes et subtiles :

1. TPCRT supporte toutes les pages vidéo sur les adaptateurs couleurs et non seulement la page zéro. Si vous changez les pages vidéo en appelant *InitPageVisible*, vous pouvez continuer à utiliser *Read, Write, GotoXY* et ainsi de suite. CRT ne vous le permet pas.

2. Contrairement à CRT, l'unité TPCRT est beaucoup plus dépendante de certaines variables globales : *PageActuelle, ModeActuel, LargeurEcran, HauteurEcran, AffichageActuel* et *SegmentVideo*. Alors que CRT calcule constamment l'adresse de départ de la mémoire vidéo, par exemple, TPCRT suppose que la valeur contenue dans *SegmentVideo* est correcte. Cela permet de gagner du temps mais vous devrez utiliser les routines TPCRT pour changer les modes vidéo, changer les pages d'affichage ou changer de mode (43 lignes/50 lignes). Si vous utilisez une autre routine (une routine d'un programme graphique par exemple), vous devrez appeler *LitModeVideo* (Réinitialise *ModeActuel, LargeurEcran, HauteurEcran, PageActuelle, AffichageActuel, SegmentVideo*) ou *ReinitVideo* (réinitialise également *CheckSnow* et les coordonnées de la fenêtre actuelle). Aussi, si vous écrivez un programme résident qui utilise TPCRT, vous devrez appeler l'une de ces deux routines, en général *ReinitVideo*.

3. Toutes les routines d'entrée du clavier sont écrites pour fonctionner avec des programmes résidents. Si vous écrivez un programme résident qui utilise les routines d'entrée de l'unité standard CRT (*ReadKey, Read, Readln*) vous bloquerez fréquemment les autres programmes résidents : SideKick ne pourra pas se charger au-dessus de votre programme TSR pendant l'attente de la frappe d'une touche. TPCRT permet d'éviter ce problème.

4. TPCRT restaure la variable globale *LongBuf* disponible en Turbo 3 et maintenant en Turbo 4. Elle vous permet de limiter la longueur des chaînes de caractères lues avec les instructions *Read* ou *ReadLn*. Par exemple, si vous devez limiter *MaChaine* à dix caractères :

```
LongBuf := 10;
Readln(MaChaine);
```

La valeur par défaut de *LongBuf* est de 126 caractères et celle-ci est réinitialisée après chaque instruction *Read* ou *ReadLn*.

5. Le code d'initialisation de TPCRT reconnait automatiquement les matériels Compaq et Zénith (seulement certains modèles) dont les adaptateurs vidéo n'exigent pas de détection de parasites et règle l'indicateur *CheckSnow* en conséquence. Ceci permet d'améliorer considérablement la performance de ces machines.

6. TPCRT supporte les systèmes multi-tâches tels que DESKVIEW et OMNIVIEW.

Evidemment, ces différences mineures ne sont pas suffisantes pour vous inciter à utiliser TPCRT plutôt que CRT. Les raisons majeures pour utiliser TPCRT se trouvent dans les nombreuses extensions qu'elle possède. La plupart sont regroupées par type : gestion du curseur, gestion de fenêtres, écriture et lecture d'écrans, gestion du clavier.

Le groupe le plus complet est celui relatif à la gestion du curseur. TPCRT vous permet de modifier la forme du curseur avec les routines *CursNormal*, *CursPlat*, *CursBloc* et *CursCache*. Vous trouverez également d'autres versions des routines *GotoXY* ou *WhereX* ou *WhereY* qui utilisent les coordonnées écran absolues plutôt que les coordonnées relatives. (Par exemple, par rapport à la fenêtre actuelle). Les routines les plus faciles à utiliser sont certainement *LitEtatCurs* et *RestaureEtatCurs* qui permettent facilement de sauvegarder et de restaurer la position et la taille actuelle du curseur.

Les autres groupes de routines ont des fonctions particulières. *EcritVite* permet d'écrire rapidement une chaîne de caractères, en vous donnant le contrôle complet sur le positionnement de la chaîne et les couleurs utilisées. *DeplaceEcran* déplace rapidement des blocs de données vers la mémoire vidéo ou de la mémoire vidéo vers l'écran sans parasites (dus à l'écriture mémoire vidéo à un instant inapproprié) ni scintillements (dus à l'activation ou à la désactivation du balayage vidéo). *SauveFen* et *RestaureFen* qui utilisent *DeplaceEcran*, permettent d'utiliser des fenêtres "popup" facilement et *CadreFen* permet de dessiner des cadres autour de ces fenêtres.

Compatibilité avec CRT

TPCRT, bien que compatible avec CRT, ne pourra la remplacer tant que les unités standard GRAPH3 et TURBO3 seront indispensables. TPCRT et CRT entrerons en conflit si vous tentez de les utiliser ensemble dans un même programme. Pour éviter ces conflits de contrôle de l'écran, TPCRT vérifie si l'unité CRT est présente et arrête le programme dans ce cas.

En plus du contrôle initial, plusieurs contrôles sont effectués lorsque certaines procédures clés sont appelées (dans le cas où CRT est initialisé après TPCRT).

Si vous utilisez GRAPH3 ou TURBO3, vous devez acheter le code source de la bibliothèque *Turbo Pascal* 5.0 , modifiez les instructions USES pour que les deux unités appellent TPCRT au lieu de CRT et recompilez-les.

Les mêmes problèmes se posent si vous tentez d'utiliser TPCRT avec une unité non standard qui utilise CRT. Cela pose moins de problème qu'avec GRAPH3 et TURBO3, puisque vous disposez du code source de l'unité et qu'il vous suffit simplement de remplacer CRT par TPCRT dans l'instruction USES. Vous pouvez néanmoins rencontrer des difficultés avec des unités précompilées en format TPU qui proviennent du domaine public ou de produits commercialisés (en général, les unités du domaine public contiennent le code source).

L'expérience a montré que l'unité qui pose le plus de problèmes de ce point de vue est l'unité TACCESS de la *Turbo Database Toolbox*. Pour y remédier, modifiez simplement l'instruction USES dans TACCESS.PAS au cas où vous ne connaîtriez pas les unités qui utilisent CRT. Vous pouvez utiliser l'utilitaire GREP de Borland pour rechercher les occurrences de "CRT" dans tous vos fichiers sources. Veuillez taper en conséquence la ligne suivante :

```
GREP -r- -w+ "CRT" *.pas
```

Note importante : Pour les utilisateurs des premières versions de *Turbo Professionnel* 4.0, TPCRT n'est pas incompatible avec l'unité standard GRAPH comme nous l'avions indiqué initialement.

Note importante : Pour tous les utilisateurs de *Turbo Professionnel* 4.0, les variables globales *LargeurActuelle* et *HauteurActuelle* sont encore supportées mais ne sont plus documentées. Nous vous recommandons d'utiliser à la place *LargeurEcran* et *HauteurEcran*. *LargeurEcran* contient la largeur actuelle de l'écran comme *LargeurActuelle*. *HauteurEcran* contient la *HauteurActuelle* de l'écran (25,43,50) alors que *HauteurActuelle* contient la *HauteurActuelle* moins 1 (24,42,49).

Constantes, variables et routines appartenant également à l'unité Turbo CRT

En général, les constantes, les variables et les routines qui suivent sont fonctionnellement identiques à celles de l'unité CRT standard de Turbo. Seuls quelques points particuliers méritent d'être notés :

1. Bien que la routine *TextMode* de TPCRT réinitialise certaines variables globales importantes (celles initialisées par *LitModeVideo*), elle n'initialise pas toutes les variables. Par conséquent, après le changement de mode vidéo, vous devez appeler *ReinitVideo* pour être

sûr qu'elles sont toutes initialisées correctement. La routine *TextMode* appartenant à TPCRT diffère également de la version standard car elle vous permet de basculer en mode graphique. (Remarquez que les supports graphiques avec TPCRT sont généralement très limités).

2. La routine *DirectVideo* a pratiquement le même rôle que celle de l'unité Turbo CRT. Elle définit la façon dont les routines *Write/WriteLn*, *Read/ReadLn* et *GotoXY* affichent le texte sur l'écran et la façon dont elles déplacent le curseur. Elle ne doit pas, quoi qu'il en soit, modifier les routines d'écriture d'écran non standard de TPCRT telles que *EcritVite* et *EcritFacile* qui écrivent toujours directement en mémoire vidéo.

ClrScr permet de remplir une fenêtre ou un écran avec un caractère différent de l'espace. Par exemple, pour remplir l'écran avec des astérisques, initialisez *CaracTexte* avec '*' avant d'appeler *ClrScr*. *CaracTexte* est automatiquement remis à ' ' lorsque l'écran a été effacé.

Constantes

```
Black = 0;
Blue = 1;
Green = 2;
Cyan = 3;
Red = 4;
Magenta = 5;
Brown = 6;
LightGray = 7;
DarkGray = 8;
LightBlue = 9;
LightGreen = 10;
LightCyan = 11;
LightRed = 12;
LightMagenta = 13;
Yellow = 14;
White = 15;
Blink = 128;
BW40 = 0;
CO40 = 1;
C40 = CO40;
BW80 = 2;
CO80 = 3;
C80 = CO80;
Mono = 7;
Font8x8 = 256;
```

Variables

```
CheckBreak= Boolean;        {active le contrôle de Ctrl-Break}
CheckEOF : Boolean;         {active le contrôle de Ctrl-Z}
CheckSnow : Boolean;        {initialisée à "True" pour empêcher la
                             neige sur les adaptateurs CGA}
```

```
DirectVideo: Boolean;        {écrit directement à l'écran}
LastMode : Word;             {mode vidéo actuel dans l'octet de poids
                             faible et l'indicateur 8*8 dans l'octet
                             de poids fort}
SaveInt1B : Pointer;         {INT $1B précédente}
TextAttr : Bytes;            {attribut vidéo actuel}
WindMin : Word;      {Xbas et Ybas de la fenêtre : dans 0..24, 0..79}
WindMax : Word;      {Xhaut et Yhaut de la fenêtre: dans 0..24, 0..79}

procedure AssignCrt (var F:Text);
   {déroute les entrées/sorties écrans standard sur TPCRT plutôt que
    vers le DOS}

procedure ClrEol;
   {efface ce qui reste de la ligne d'écran en cours}

procedure ClrScr;
   {efface la fenêtre actuelle}

procedure Delay(MS : Word);
   {pause de MS millisecondes}

procedure DelLine;
   {efface la ligne d'écran en cours}

procedure GotoXY(X, Y : Byte);
   {déplace le curseur en colonnes X ligne Y en coordonnées relatives
    dans la fenêtre}

procedure HighVideo;
   {initialise le bit d'intensité dans TextAttr}

procedure InsLine;
   {insère une nouvelle ligne à la position du curseur}

procedure LowVideo;
   {efface le bit d'intensité dans TextAttr}

function KeyPressed : boolean;
   {renvoie "True" si une touche a été pressée}

procedure NormVideo;
   {initialise TextAttr avec l'attribut vidéo utilisé au démarrage du
    programme (non équivalente à  HighVideo!)}

procedure NoSound;
   {désactive le son}

function ReadKey : char;
   {lit un caractère à partir du clavier}

procedure Sound(HZ : Word);
   {active le son avec la fréquence désignée}

procedure TextBackground(couleur : byte);
   {initialise la couleur de fond pour les écritures écran}

procedure TextColor(couleur : byte);
   {initialise la couleur de l'avant-plan pour les écritures écran}
```

```
procedure TextMode(mode : Word);
   {initialise le mode vidéo actuel}

function WhereX : byte;
   {renvoie la coordonnée actuelle X (colonne) du curseur en
    coordonnées relatives dans la fenêtre}

function WhereY : byte;
   {renvoie la coordonnée actuelle Y (ligne) du curseur en
    coordonnées relatives dans la fenêtre}

procedure Window(Xbas, Ybas, Xhaut, Yhaut : byte);
   {initialise les coordonnées de la fenêtre actuelle;}
```

TPCRT : déclarations

Type

```
NomRe = array[1..11] of char;
```

Représentation interne du nom associé à un répertoire d'entrée dans une bibliothèque de fenêtres. Le nom est sur les 8 premiers caractères, le suffixe sur les trois derniers.

```
EntRepertoire=
Record
   Statut : ShortInt; {0 = utiliser, -1 = inutiliser, -2 = supprimer}
   Nom : NomRe;              { nom et suffixe en format FCB}
   Index : Word;             { index de cet enregistrement}
   LongMemb : Word;          {nombre de blocs de 128 octets utilisés par
                              cet  enregistrement}
   CRC : Word;               {non implémenté}
   DateCreation : Word;      {date de création -- non implémentée}
   DateDernModif : Word;     {date de la dernière modification - non
                              implémentée}
   HeureCreation : Word;     {heure de la création - non implémentée}
   HeureDernModif : Word;    {heure de la dernière modification non
                              implémentée}
   CompteRemplissage : Byte;        {octet inutilisé dans le dernier
                                     bloc de  l'enregistrement}
   Filler : array[28..32] of byte;         {remplissage sur 32 octets}
end;
```

Format d'une entrée de répertoire dans la bibliothèque créée avec *CreeBiblio*.

```
TypeRepertoire = array[0..255] of EntRepertoire;
PtrRepertoire = ^TypeRepertoire;
```

Format du répertoire en tête de la bibliothèque créée avec *CreeBiblio*. Le nombre réel d'entrées dans le répertoire peut varier.

```
TypeAffichage = (MonoHerc, CGA, MCGA, EGA, VGA, PGC);
```

Les 6 types standard d'adaptateurs vidéo reconnus par TPCRT. MonoHerc signifie soit un adaptateur monochrome ou une carte graphique monochrome Hercule. PGC correspond à une carte contrôleur graphique professionnel IBM.

```
AttrsFaciles = array[0..3] of byte;
```

Tableau des attributs utilisés par la procédure *EcritFacile*.

```
TypeCaracCadre = (Hgauche, Bgauche, Hdroit, Bdroit, Horiz, Verti);
TabCadre = array[TypeCaracCadre] Of Char;
```

Types de caractères utilisés par *CadreFen* pour le dessin des cadres autour des fenêtres et tableau de ces caractères.

```
NomBiblio = String[12];
```

Nom d'une entrée de répertoire dans la bibliothèque (un nom de fichier).

```
EcranCompacte = array[1..4000] of byte;
```

Contient l'image écran compactée avec la procédure *CompacteFen*. La taille réelle du tableau est variable.

```
FenCompactee =
   record
      Taille : Word;          {taille d'une fenêtre compactée y compris
                               cet entête}
      LigneSup : Byte;        {coordonnées du coin supérieur gauche d'une
                               fenêtre}
      ColSup : Byte;
      Lignes : Byte;          {hauteur d'une fenêtre}
      Cols : Byte;            {largeur d'une fenêtre}
      DebA : Word;            {index du début de la section des attributs
                               dans Contenu}
      CDelta : Word;          {nombre d'octets avant les premiers
                               caractères - EnrCompacte}
      ADelta : Word;          {nombre d'octets avant les premiers Attr -
                               EnrCompacte}
      Contenu : EcranCompacte;     {contenu de l'écran compacté}
   end;
PtrFenCompactee = ^FenCompactee;
```

Format de l'écran compacté avec la procédure *CompacteFen*.

Contenu, qui contient l'écran compacte lui-même, est un tableau de *Taille* octets, moins la taille des 8 premiers champs de l'enregistrement (12 octets).

```
CoordFen =
   record
      XB, YB, XH, YH : byte;
   end;
```

Enregistrements utilisés pour sauvegarder/restaurer les coordonnées d'une fenêtre.

Constantes

```
BiosScroll : boolean = True;
```

Si cette constante vaut "False", une routine spéciale est utilisée pour supprimer la neige et le scintillement avec les adaptateurs graphiques couleurs. Cette valeur affecte non seulement les instructions *Write* et *Writeln* mais aussi les routines *DerouleFenHaut* et *DerouleFenBas*. La routine de déroulement du BIOS est normalement plus rapide.

```
ErreurCrt : Word = 0;
```

Mieux adaptée que la variable *DOSError* de l'unité DOS, cette variable renvoie les codes d'erreur d'entrées/sorties des routines de TPCRT relatives aux entrées/sorties fichier. (*CreeBiblio, OuvreBiblio,* etc).

```
CrtMemInsuf = $FFFF;
CrtPointeurNil = $FFFE;
CrtBiblioNonValide = $FFFD;
CrtFenAbsente = $FFFC;
CrtDirPlein = $FFFB;
CrtFenIncorrecte  = $FFFA;
```

Ces constantes représentent les codes d'erreurs non standard que vous pourriez trouver dans *ErreurCrt*. Si *ErreurCrt* vaut 0, et n'appartient pas à l'intervalle *CrtFenIncorrecte..CrtMemInsuf*, la valeur retournée est le code d'erreur standard de la fonction IoResult.

```
DetecteMultitache : Boolean = False;
```

Si *DetecteMultitache* vaut "True", TPCRT effectuera les contrôles nécessaires pour déterminer si le programme fonctionne sous un système multi-tâches tel que *DesqView* ou *OmniView* (initialement *TaskView*). Si c'est le cas, *MultitacheOn* prendra la valeur "True" et toutes les écritures écran (qui utilisent *Write, WriteLn, EcritRapide* etc...) seront dirigées vers l'écran virtuel fourni par le système plutôt que vers l'écran physique. Comme la valeur par défaut de *DetecteMultitache* est "False", vous devrez ajouter deux lignes en tête de votre programme si vous voulez qu'il puisse s'exécuter comme une tâche d'arrière plan :

```
DetecteMultitache : = True;
ReinitVideo;
```

Si le programme utilise TPECRAN appelez *ReinitEcran,* plutôt que *ReinitVideo*. Cette caractéristique permet à la plupart des programmes écrits avec TPCRT et TPECRAN (et par conséquent TPFENETR, TPMENU, TPEDIT et TPSAISIE) d'être exécutés en tâche d'arrière-plan à l'intérieur d'une fenêtre (les seules exceptions que nous connaissons sont les programmes qui utilisent les autres pages vidéo). Remarquez que TPCRT ne supporte pas plus *TopView* qu'il ne supporte *Windows* de Microsoft ou *DoubleDOS*.

Note importante : Ne positionnez pas *DetecteMultiTache* à "True" dans un programme TSR. L'appel de la fonction utilisée pour déterminer si un système multi-tâches est présent, peut causer certaines difficultés si elle est exécutée à l'intérieur d'une partition qui a été définie comme "écrivant directement sur écran". C'est pourquoi la valeur par défaut de *DetecteMultitache* est "False". Si vous désirez écrire des routines TSR, vous pouvez modifier la valeur par défaut et lui donner la valeur "True" puis recompiler TPCRT.

```
InhibeAff : ShortInt = -1;
```

Le mécanisme d'auto-détection des adaptateurs vidéo étant imparfait, nous vous donnons maintenant le moyen d'y remédier : *InhibeAff*. Cette variable est initialisée à -1, pour activer l'auto-détection. Si vous devez forcer TPCRT pour traiter un adaptateur EGA ou CGA, vous pouvez initialiser *InhibeAff* avec une valeur ordinale correspondant au type *TypeAffichage* -- "*InhibeAff := Ord(CGA);*"-- puis appeler *ReinitVideo*. L'utilisation la plus courante de cette variable est de forcer TPCRT à reconnaître un adaptateur EGA ou VGA si l'auto-détection a mal fonctionné. Cela sera rarement nécessaire.

```
CaracsCadre : TabCadre = #213#212#184#190#205#179;
```

Sert à enregistrer les caractères utilisés par *CadreFen* pour dessiner des cadres autour des fenêtres. Elle peut être modifiée directement ou indirectement (en utilisant *InitCaracsCadre*).

```
FixeCouleurs : boolean = True;
```

Si cette constante vaut "True", la routine *FixeCouleur déterminera* les couleurs (comme cela est fait dans TPECRAN) pour améliorer l'affichage. La routine *FixeCouleur* est utilisée par *InitFen, EcritViteFen* (TPCRT) et *SousMenu* (TPMENU). Les valeurs données à *FixeCouleurs* ne modifient pas le comportement des routines appartenant à l'unité TPCRT, sauf la routine *FixeCouleur* (excepté dans les cas explicitement détaillés plus loin dans le chapitre), et n'affecte pas le fonctionnement des routines de l'unité TPECRAN.

```
CaracTexte : char = ' ';
```

Ainsi avec *TextAttr*, ce caractère est utilisé par *ClrScr, ClrEol, DerouleFenHaut* et *DerouleFenBas*, lors de l'effacement ou du déroulement d'une fenêtre. Elle est automatiquement réinitialisée à sa valeur par défaut ' ', lorsque l'écran a été effacé ou déroulé.

Variables

```
LongBuf : Word;
```

Longueur maximum d'une chaîne de caractères pour *Read/ReadLn*. L'intervalle valide est 1..126 et *LongBuf* est toujours réinitialisée à 126 après chaque opération *Read*.

```
IndicCtrlBreak : boolean;
```

A la valeur "True" lorsque (CTRL)-(BREAK) a été pressé. Utilisé en interne seulement. Cette variable est exportée seulement pour que TPECRAN puisse y accéder.

```
AffichageActuel : TypeAffichage;
```

Adaptateur d'affichage sélectionné. Si deux adaptateurs d'affichage sont installés, *AffichageActuel* vous indique seulement le type de l'adaptateur actif. Voir également la routine *AfficheMeilleur*.

```
ModeActuel : Byte Absolute LastMode;
```

Mode vidéo actuel.

```
PageActuelle : Byte;
```

Page vidéo actuelle.

```
AfficheMeilleur : TypeAffichage;
```

Cette variable est significative seulement si elle vaut MCGA, VGA, ou EGA. Elle sert habituellement à repérer deux adaptateurs, un adaptateur monochrome et un adaptateur EGA connecté à un moniteur couleur par exemple. Dans de tels cas, la routine *AfficheMeilleur* aura la valeur EGA même si *AffichageActuel* vaut *MonoHerc*. Cette information est surtout utile si des programmes résidents en mémoire doivent supporter les modes 43 et 50 lignes (qui utilisent la police 8x8) disponibles sur les adaptateurs EGA et VGA respectivement. Voir SMACS.PAS pour un exemple. Si un adaptateur MCGA, VGA ou EGA n'est pas installé, *AfficheMeilleur* sera positionnée à *MonoHerc*.

```
EnModeTexte : boolean;
```

Initialisé à "False" en mode graphique. (Remarquez que le mode graphique Hercule n'est pas détecté).

```
MultiTacheOn : Boolean;
```

Cette variable est initialisée à "True" si *DetecteMultitache* vaut "True" et qu'un système multi-tâches (*DesqView*, *TaskView*/*OmniView*) est présent.

```
AttrNormal : Byte;
```

Attribut vidéo pour vidéo normale.

```
UneMS : Word;
```

Cette variable contient le compteur de boucle utilisé par Delay pour générer une attente d'une mili-seconde.

```
SaveInt1B : Pointer;
```

Le précédent gestionnaire d'interruptions $1B (CRT et TPCRT utilisent cette interruption afin de trapper (CTRL)-(BREAK)). Cette variable existe seulement dans TPCRT si elle a été compilée avec *Turbo Pascal* 4.0. En *Turbo Pascal* 5.0, *SaveInt1B* se trouve dans l'unité SYSTEM.

```
HauteurEcran : Word;
```

Hauteur actuelle de l'écran affiché (habituellement 25).

```
LargeurEcran : Word;
```

Largeur actuelle de l'écran affiché.

```
SegmentVideo : Word;
```

Segment de l'adresse de début de la mémoire vidéo. Normalement, $B000 pour un adaptateur monochrome ou $B800 pour un adaptateur couleur ; d'autres valeurs sont possibles si *PageActuelle* est différente de zéro.

```
SegVirtuel : Word;
```

Normalement, cette variable est égale à *SegmentVideo* mais elle est quelquefois modifiée par TPFENETR pour permettre à *EcritRapide* et à d'autres routines qui écrivent sur l'écran d'être utilisées sur des écrans virtuels. Consultez la documentation de TPFENETR.

```
LargVirt : Word;
```

Cette variable est normalement égale à *LargeurEcran,* mais elle est quelquefois modifiée par TPFENETR pour permettre à *EcritRapide* et à d'autres routine d'écriture sur écran d'être utilisées sur des écrans virtuels. Consultez la documentation de TPFENETR.

AdapteCouleursFenCompactee

Déclaration

```
procedure AdapteCouleursFenCompactee(PFC : PrtFenCompactee);
```

Objet

Adapte les couleurs d'une fenêtre compactée pour améliorer l'affichage en Mono/N&B.

Commentaires

Cette routine adapte (en utilisant *FixeCouleur*) tous les attributs vidéo d'une fenêtre packée créée par *CompacteFen*. En général, elle est utilisée lorsqu'une fenêtre est dessinée sur un écran couleur, sauvegardée sur un fichier, puis relue à partir d'un disque (en utilisant *LitFenCompactee* ou *LitFenBiblio*) sur une autre machine.

Remarquez que l'appel permanent de cette procédure modifie les attributs de la fenêtre compactée, aussi si vous sauvegardez à nouveau la fenêtre sur un fichier après avoir réadapté les couleurs, les informations de couleurs initiales seront perdues.

Exemple

```
PFC := LitFenCompactee('Fenetre.dat');
AdapteCouleursFenCompactee(PFC);
AffFenCompactee(PFC);
```

Lit la fenêtre compactée rangée dans FENETRE.DAT en mémoire, adapte les couleurs, puis l'affiche.

Voir aussi

FixeCouleur (TPCRT)
CompacteFen (TPCRT)
LitFenCompactee (TPCRT)
LitFenBiblio (TPCRT)

AffFenCompactee

Déclaration

```
procedure AffFenCompactee(PFC : PtrFenCompactee);
```

Objet

Affiche la fenêtre compactée pointée par *PFC*.

Commentaires

Cette routine affiche une fenêtre compactée, précédemment créée avec *CompacteFen*. Remarquez qu'il n'est pas nécessaire d'indiquer l'emplacement de la fenêtre à afficher puisque cette information a été sauvegardée par *CompacteFen* avec le contenu de la fenêtre.

Exemple

```
PFC := CompacteFen(1, 1, 80, 25);
...
AffFenCompactee(PFC);
```

Sauvegarde l'écran comme une fenêtre compactée, puis réaffiche la fenêtre compactée.

Voir aussi

CompacteFen (TPCRT)

AffFenCompacteeEn

Déclaration

```
procedure AffFenCompacteeEn(PFC : PtrFenCompactee; Ligne, Col : Byte);
```

Objet

Affiche la fenêtre compactée pointée par *PFC* en *Ligne, Col.*

Commentaires

Cette routine affiche le fenêtre compactée pointée par *PFC* en *Ligne* et *Col.* Si nécessaire, les coordonnées spécifiées sont ajustées pour que la fenêtre puisse tenir sur l'écran. Pour une fenêtre de 40 colonnes avec *Col* = 70, *Col* prendra la valeur 41 (avec un écran 80 colonnes) pour cadrer correctement la fenêtre.

Exemple

```
PFC := CompacteFen(1, 20, 80, 25);
...
AffFenCompacteeEn(PFC, 1, 1);
```

Sauvegarde les six dernières lignes de l'écran comme une fenêtre compactée puis les affiche sur les 6 premières lignes de l'écran.

AjoutFenBiblio

Déclaration

```
Procedure AjoutFenBiblio(PFC : PtrFenCompactee ; var F : file;
                         PR : PtrRepertoire ; NomFen : NomBiblio);
```

Objet

Ajoute une fenêtre compactée à une bibliothèque donnée.

Commentaires

Cette routine ajoute une fenêtre compactée (PFC^) créée avec *CompacteFen* à la bibliothèque (F) déjà ouverte par *OuvreBiblio* ou *CreeBiblio*. *PR* pointe sur le répertoire de la bibliothèque. *NomFen* est le nom affecté à la fenêtre ; c'est-à-dire un nom de fichier d'au plus 8 caractères avec un suffixe facultatif d'au plus 3 caractères.

Si la bibliothèque contient déjà une entrée de même nom, cette entrée sera supprimée logiquement et la nouvelle fenêtre sera créée en fin de bibliothèque.

A la sortie, *ErreurCrt* contient le code d'erreur éventuel. Outre les codes d'erreurs d'entrées/sorties habituels, la seule valeur possible pour *ErreurCrt* est *CrtRepPlein* (répertoire de la bibliothèque plein).

```
PFC := CompacteFen(1, 1, 80, 25);
PR := OuvreBiblio(MaBib, 'MABIB.BIB');
AjoutFenBiblio(PFC, MaBib, PR, 'MAFEN.1');
```

Compacte l'écran puis le sauvegarde sous le nom MAFEN.1 dans MALIB.BIB

Voir aussi

CreeBiblio (TPCRT)
OuvreBiblio (TPCRT)
CompacteFen (TPCRT)

CadreDroite

Déclaration

```
procedure CadreDroite(Ch : string; Ligne, Attr : Byte);
```

Objet

Ecrit *Ch* cadré à droite sur la ligne de la fenêtre avec l'attribut *Attr*.

Commentaires

Cette routine affiche la chaîne spécifiée (*Ch*) cadrée à droite de la fenêtre actuelle, sur la ligne spécifiée avec l'attribut vidéo donné (Attr). Le numéro de ligne est relatif à la fenêtre actuelle.

Exemple

```
Window(2, 2, 79, 24);
CadreDroite('Bonjour à tous !', 1, $F);
```

Affichera 'Bonjour à tous !' en ligne 2 (absolu) avec la fin de chaîne ('!') touchant le bord de la fenêtre en colonne 79.

Voir aussi

CentreVite (TPCRT)

CadreFen

Déclaration

```
procedure CadreFen(Colg, LigneSup, Cold, LigneInf : byte;
                   AttrC, AttrE : byte;
                   entete : string );
```

Objet

Dessine un cadre autour d'une fenêtre.

Commentaire

Cette routine dessine un cadre (ou boîte) autour d'une fenêtre dont les dimensions sont définies par *Colg, LigneSup, Cold* et *LigneInf.* (Remarquez que ces coordonnées d'écran sont traitées en mode absolu et non en mode relatif dans la fenêtre actuelle). Les caractères utilisés pour dessiner le cadre sont enregistrés dans la constante typée globale *CaracsCadre* qui peut être modifiée directement ou indirectement (en utilisant *InitCaracsCadre*). L'entête est une chaîne à afficher au milieu de la ligne supérieure de la fenêtre. Si aucun entête n'est désiré, vous pouvez passer une chaîne vide. *AttrC* est l'attribut vidéo utilisé pour afficher les caractères du cadre et *AttrE* est l'attribut vidéo pour l'entête.

Exemple

```
CadreFen(1, 1, LargeurEcran, HauteurEcran, TextAttr, TextAttr,
   ' L''entête se place ici ');
```

Dessine un cadre autour de l'écran.

Voir aussi

InitFen (TPFENETR) InitCaracsCadre (TPCRT).

CentreVite

Déclaration

```
procedure CentreVite(CH : string; Ligne, Att : Byte);
```

Objet

Ecrit la chaîne *Ch* centrée dans ligne de la fenêtre avec *Attr* comme attribut vidéo.

Commentaires

Cette routine affiche la chaîne spécifiée (*Ch*) centrée à l'intérieur de la fenêtre actuelle sur la ligne spécifiée avec l'attribut vidéo *Attr*. *Ligne* est relative à la fenêtre actuelle.

Exemple

```
Window(2, 2, 79, 24);
CentreVite('Bonjour à tous !', 1, $F);
```

Affichera 'Bonjour à tous !' en ligne 2, (absolu), centré entre les colonnes 2 et 79.

Voir aussi

CadreDroite (TPCRT)

ChangeAttr

Déclaration

```
procedure ChangeAttr(Nombre : Word; Ligne, Col, Attr : Byte );
```

Objet

Modifie le nombre des attributs vidéo avec *Attr* en commençant en position *Ligne, Col.*

Commentaires

Cette routine vous permet de modifier les couleurs de l'écran sans modifier le texte de l'écran. L'utilisation la plus typique de *ChangeAttr* est probablement le déplacement de la barre lumineuse sur les menus puisqu'elle vous permet de mettre en valeur (ou non) une partie déterminée de l'écran sans savoir ce qui a été mis en valeur.

Nombre spécifie le nombre des attributs vidéo à modifier. *Ligne* et *Col* indiquent les coordonnées du début de la partie à modifier (remarquez que ces coordonnées d'écran sont en mode absolu et non en mode relatif). *Attr* spécifie le nouvel attribut vidéo pour la partie concernée.

Pour une explication rapide des attributs vidéo : voir la documentation de REF au chapitre 13.

Exemple

```
program Couleurs;
uses TpCrt;
   const Attr : Byte = 0;
begin
   ClrScr;
   WriteLn('     Bonjour à tous !     '^M^J'Observez les changements
           de couleur...');
   Delay(1000);
   while not KeyPressed do begin
      ChangeAttr(26, 1, 1, Attr and $7F);
               {Supprime le bit de clignotement}
      Delay(50); {le temps de regarder...}
      Inc(Attr); {change l'attribut}
   end;
end.
```

Balaie rapidement toutes les combinaisons de couleurs jusqu'à ce qu'une touche soit pressée.

Voir aussi

EcritVite (TPCRT)

ChangeAttrFen

Déclaration

```
Procedure ChangeAttrFen(Nombre : Word; Ligne, Col, Attr : Byte);
```

Objet

Modifie *Nombre* attributs vidéo en commençant en *Ligne, Colonne,* avec *Attr*.

Commentaires

Cette routine se superpose à *ChangeAttr*. Elle modifie les coordonnées relatives de la fenêtre en coordonnées absolues, puis appelle *ChangeAttr*. Si la variable globale *FixeCouleurs* vaut "True", l'attribut vidéo Attr est également adapté avant d'appeler *ChangeAttr*.

Remarquez qu'aucun contrôle de limite n'est effectué par *ChangeAttrFen*. Ceci suppose que les coordonnées de fenêtre passées en paramètre sont valides et que tous les attributs qui doivent être changés, appartiennent à la fenêtre actuelle.

CompacteBiblio

Déclaration

```
procedure CompacteBiblio(NomB : string);
```

Objet

Compacte une bibliothèque pour supprimer physiquement les entrées effacées logiquement.

Commentaires

Cette routine compacte une bibliothèque crée par *CreeBiblio* pour supprimer physiquement toutes les entrées qui ont été effacées logiquement par *SupprimeFenBiblio*. NomB est le nom de la bibliothèque à compacter qui doit être fermée après l'appel à *CompacteBiblio*. Si une erreur se produit, *ErreurCrt* contient le code d'erreur. Outre les codes d'erreurs d'entrée/sortie habituels, les seules valeurs possibles pour *ErreurCrt* sont *CrtBiblioNonValide* (le fichier est incorrect) ou *CrtMemInsuf* (Pas assez de place mémoire pour allouer le répertoire).

Lors de la compression de la bibliothèque, *CompacteBiblio* crée un fichier temporaire appelé '$$WORK$$.LBR', recopie toutes les entrées non effacées logiquement dans le fichier temporaire, efface NomB puis, renomme le fichier temporaire.

Exemple

```
PR := OuvreBiblio(MaBib, 'MABIB.BIB');
SupprimeFenBiblio(MaBib, PR, 'MAFEN.1');
FermeBiblio(MaBib, PR);
CompacteBiblio('MABIB.BIB');
```

Supprime logiquement MAFEN.1 de MABIB.BIB, puis compacte la bibliothèque pour supprimer physiquement la fenêtre.

Voir aussi

CreeBiblio (TPCRT)

SupprimeFenBiblio (TPCRT)

CompacteFen

Déclaration

```
function CompacteFen(XBas, YBas, XHaut, YHaut : byte)
                    : PrtFenCompactee;
```

Objet

Renvoie un pointeur sur une fenêtre compactée ou nil s'il n'y pas assez de place mémoire.

Commentaires

Cette routine est semblable à la routine *SauveFen* mais enregistre le contenu de la fenêtre désignée en format compacté. Plus précisément, elle enregistre le contenu de la fenêtre comme un couple de tableaux, l'un contenant les caractères, l'autre contenant les attributs, et compacte chacun des deux tableaux. Cette technique permet de convertir toute suite de six octets, ou plus, identiques en un enregistrement de cinq octets. Le gain obtenu est en général compris entre 30 et 90 %.

Les dimensions de la fenêtre sauvegardée sont spécifiées par *XBas*, *YBas*, *XHaut* et *YHaut*. Ces dimensions et d'autres données sur la fenêtre sont sauvegardées avec le contenu de la fenêtre si bien que lorsque vous voulez réafficher la fenêtre (avec *AffFenCompactee*), vous n'avez pas besoin de spécifier les coordonnées de la fenêtre une deuxième fois. *CompacteFen* renvoie nil s'il n'y a pas assez d'espace mémoire pour compacter la fenêtre et un pointeur sur la fenêtre compactée sinon. La mémoire allouée à la fenêtre peut être récupérée en utilisant *LibereFenCompactee*.

TPCRT contient un ensemble de routines pour afficher les fenêtre compactées, les sauvegarder sur des fichiers, les relire sur disque et pour travailler avec des bibliothèque de fenêtres compactées.

Exemple

```
PFC := CompacteFen(1, 1, 80, 25);
...
AffFenCompactee(PFC);
LibereFenCompactee(PFC);
```

Sauvegarde la fenêtre actuelle en format compacté, la réaffiche puis, la libère.

Voir aussi

CreeBiblio (TPCRT)
AffFenCompactee (TPCRT)
LibereFenCompactee (TPCRT)
SauveFen (TPCRT)

ContrlClavier

Déclaration

```
function ContrlClavier(Var CodeTouche : Word ) : boolean;
```

Objet

Renvoie "True" (et le code touche) si une touche est en attente.

Commentaires

Cette routine est semblable à *KeyPressed* mais elle renvoie également le code de la touche si une touche est en attente. Elle permet donc de connaître la touche en attente sans avoir à appeler *ReadKey* ou *LitMotTouche* qui effacent la touche dans le buffer du BIOS.

Exemple

```
var
   CodeTche : Word;
...
if ContrlClavier(CodeTche) then
   if Lo(CodeTche) = 27 then begin
      CodeTche := LitMotTouche;
      Exit;
   end;
```

Contrôle si une touche est en attente. Si (ESC) est pressée, lit la touche et rend le contrôle. Sinon, le traitement se poursuit.

Voir aussi

LitMotTouche (TPCRT)

CoordXAbs

Déclaration

```
function CoordXAbs : byte;
```

Objet

Envoie la coordonnée absolue *X* du curseur (colonnes).

Commentaire

Comme la fonction standard *WhereX*, cette routine renvoie la coordonnée *X* actuelle (colonne) du curseur mais ce nombre est une coordonnée absolue et non une coordonnée relative à la fenêtre actuelle. Comme avec *WhereX*, toutes les coordonnées *X* renvoyées ont pour origine le coin supérieur gauche de l'écran (1,1).

Exemple

```
Window(3, 3, 80, 25);
GotoXY(1, 1);
```

Dans ce cas, *WhereX* retournerait 1 et *CoordXAbs* retournera 3.

Voir aussi

CoordXY (TPCRT)

CoordYAbs (TPCRT)

CoordXY

Déclaration

```
function CoordXY : Word;
```

Objet

Renvoie la coordonnée *X* absolue (colonne) et *Y* (ligne) du curseur.

Commentaire

Cette routine associe les fonctions de *CoordXAbs* et *CoordYAbs* et vous permet d'obtenir ces deux valeurs en un seul appel. Le bit de poids fort du résultat contient la coordonnée *Y* actuelle (ligne) et le bit de poids faible contient la coordonnée *X* actuelle (colonne). Comme avec *CoordXAbs* et *CoordYAbs*, toutes les coordonnées renvoyées ont pour origine le coin supérieur gauche de l'écran (1,1).

Exemple

```
Window(3, 3, 80, 25);
GotoXY(1, 1);
```

Dans ce cas, *WhereX* et *WhereY* renverront la valeur 1, *CoordXAbs* et *CoordYAbs* retourneront 3 et *CoordXY* retournera $0303 (771).

Voir aussi

CoordXAbs (TPCRT) CoordYAbs (TPCRT)

CoordYAbs

Déclaration

```
function CoordYAbs : byte;
```

Objet

Renvoie la coordonnée Y absolue (ligne) du curseur.

Commentaires

Comme la fonction standard *WhereY*, cette routine renvoie la coordonnée Y actuelle (ligne) du curseur mais le numéro est une coordonnée absolue et non une coordonnée relative à la fenêtre actuelle. Comme avec *WhereY*, toutes les coordonnées Y renvoyées ont pour origine le coin supérieur gauche de l'écran(1,1).

Exemple

```
Window(3, 3, 80, 25);
GotoXY(1, 1);
```

Dans ce cas, *WhereY* retournerait 1 et *CoordYAbs* retournera 3.

Voir aussi

CoordXAbs (TPCRT) CoordXY (TPCRT).

CreeBiblio

Déclaration

```
function CreeBiblio(var F : file; Nom :  string;
                    Entrees : Byte) : PtrRepertoire;
```

Objet

Crée une bibliothèque avec le nombre d'entrées de répertoire spécifié et renvoie un pointeur sur le répertoire.

Commentaires

Cette routine crée un fichier bibliothèque utilisé pour ranger les fenêtres compactées, créées par la procédure *CompacteFen*. Chaque bibliothèque est composée d'un répertoire est d'une suite de points d'entrées (fenêtre compactée). *F* est le fichier bibliothèque à ouvrir. *Nom* est le nom de fichier qui lui correspond. *Entrees* indique le nombre maximum de fenêtres compactées qui peuvent être rangées dans la bibliothèque. Ce paramètre doit appartenir à l'intervalle 1..255.

Puisqu'une entrée de répertoire est utilisée pour enregistrer des informations sur le répertoire lui-même, *CreeBiblio* ajoute 1 au nombre d'entrées puis arrondi le résultat au prochain multiple de 4 (l'espace du fichier est alloué en bloc de 128 octets et une entrée de répertoire occupe 32 octets ; en conséquence, le nombre réel d'entrées dans le répertoire doit être un multiple de 4).

Remarquez que les bibliothèques créées avec cette routine utilisent le format de fichier normal LBR.

Bien que ce format ait certaines faiblesses, il possède un énorme avantage : il existe un grand nombre de programmes du domaine public que vous pouvez utiliser directement pour examiner et gérer ces fichiers.

CreeBiblio renvoie nil en cas d'erreur, sinon elle renvoie un pointeur sur le répertoire de la bibliothèque qui vient d'être créée. *ErreurCrt* contiendra le code erreur. Outre les codes d'erreurs d'entrées/sorties habituels, la seule possible valeur de *ErreurCrt* est *CrtMemInsuf* (pas assez de mémoire pour allouer le répertoire).

Exemple

```
PR := CreeBiblio(MaBib, 'MABIB.BIB', 39);
```

Crée une bibliothèque (MABIB) appelée MABIB.BIB pour y enregistrer 39 entrées. *PR* pointera sur le répertoire de la bibliothèque.

Voir aussi

CompacteFen (TPCRT)

FermeBiblio (TPCRT)

CursBloc

Déclarations

```
procedure CursBloc;
```

Objet

Modifie le nombre de lignes de balayage pour obtenir un curseur en forme de pavé.

Commentaire

Cette routine modifie le nombre de lignes de balayage du curseur pour l'agrandir au maximum. Sur les machines possédant un adaptateur CGA, le curseur utilisera les huit lignes de balayage ; sur les machines possédant un adaptateur monochrome EGA ou VGA, il utilisera les quatorze lignes de balayage.

Voir aussi

CursPlat (TPCRT)
CursCache (TPCRT)
CursNormal (TPCRT)
InitTailleCurs (TPCRT)

CursCache

Déclaration

```
procedure CursCache;
```

Objet

Cache le curseur.

Commentaire

Cette routine cache le curseur physique sur n'importe quel adaptateur. Remarquez néanmoins, que le curseur peut quelques fois mystérieusement rester visible sur des systèmes possédant des adaptateurs monochromes, lorsque l'affichage est déroulé par le BIOS. C'est une bizarrerie du matériel et elle est généralement incontournable.

Voir aussi

CursBloc (TPCRT)
CursPlat (TPCRT)
CursNormal (TPCRT)
InitTailleCurs (TPCRT)

CursDebFin

Déclaration

```
function CursDebFin:Word;
```

Objet

Renvoie un mot contenant la première et la dernière lignes de balayage du curseur.

Commentaire

Cette routine combine les fonctions des routines *LigneFinCurs* et *LigneDebutCurs* en renvoyant les deux valeurs. La ligne de début est renvoyée dans l'octet de poids fort du mot, la ligne de fin dans l'octet de poids faible. Peut être utilisé en combinaison avec *InitTailleCurs* ou à la place de *LitEtatCurs* et *RestaureEtatCurs*.

Exemple

```
var
   LigBalayageCurs, Tche : Word;
begin
   LigBalayageCurs := CursDebFin;
   CursCache;
   EcritVite('Erreur. Pressez une touche pour continuer...', 25, 1,
            TextAttr);
   Tche := LitMotTouche;
   InitTailleCurs(Hi(LigBalayageCurs), Lo(LigBalayageCurs));
end.
```

Sauvegarde la taille du curseur, cache le curseur, affiche un message d'erreur en bas de l'écran, attend la frappe d'une touche puis restaure la taille du curseur.

Voir aussi

LigneFinCurs (TPCRT)
LigneDebutCurs (TPCRT)
LitEtatCurs (TPCRT)
InitTailleCurs (TPCRT)

CursNormal

Déclaration

```
procedure CursNormal;
```

Objet

Sélectionne un curseur normal avec le mode vidéo actuel.

Commentaire

Cette routine restaure le curseur à sa taille normale de deux lignes pour tous les types d'affichage.

Voir aussi

CursBloc (TPCRT)
CursPlat (TPCRT)
CursCache (TPCRT)
InitTailleCurs (TPCRT).

CursPlat

Déclaration

```
procedure CursPlat;
```

Objet

Initialise la largeur du curseur en fonction du mode vidéo actuel.

Commentaire

Cette routine modifie le nombre de lignes de balayage du curseur pour qu'il soit plus large que le curseur normal de deux lignes. Sur les machines possédant un adaptateur couleur, le curseur comportera trois des huit lignes de balayage disponibles ; sur les machines possédant un adaptateur monochrome, il comportera quatre des quatorze lignes de balayage disponibles et pour les modes EGA 43 lignes ou VGA 50 lignes, il comportera cinq des huit lignes de balayage disponibles.

Voir aussi

CursBloc (TPCRT)
CursCache (TPCRT)
CursNormal (TPCRT)
InitTailleCurs (TPCRT)

DeplaceEcran

Déclaration

```
procedure DeplaceEcran(Var Source, Dest; Longueur : Word);
```

Objet

Déplace *Longueur* mots de *Source* vers *Dest* sans scintillement.

Commentaires

DeplaceEcran est semblable à la procédure standard *Move* de Turbo mais elle est strictement conçue pour déplacer des données entre la mémoire normale et la mémoire vidéo ou d'une partie de la mémoire vidéo dans une autre. Contrairement à la routine standard *Move,* elle ne peut pas déplacer des octets élémentaires de données mais seulement des couples d'octets (mots).

Source est l'adresse de la donnée à déplacer et *Dest* l'adresse réceptrice. Si la valeur des segments d'une adresse est dans la partie de mémoire utilisée par CGA ($B8xx) et que *CheckSnow* vaut "True", *DeplaceEcran* supprimera le scintillement. Sinon elle déplacera la donnée aussi rapidement que possible. *Longueur* est le nombre de mots à déplacer.

Exemple

```
var
   PtrBufEcran : Pointer;
   OctetsSurEcran : Word;
begin
   OctetsSurEcran := (LargeurEcran * HauteurEcran) * 2;
   GetMem(PtrBufEcran, OctetsSurEcran);
   DeplaceEcran(Mem[SegmentVideo:0], PtrBufEcran^,
                    OctetsSurEcran div 2);
   {...Suite...}
   DeplaceEcran(PtrBufEcran^, Mem[SegmentVideo:0],
                    OctetsSurEcran div 2);
   FreeMem(PtrBufEcran, OctetsSurEcran);
end.
```

Sauvegarde l'écran et le restaure. Remarquez que cette tâche particulière peut être accomplie plus facilement en utilisant *SauveFen* et *RestaureFen.*

Voir aussi

RestaureFen (TPCRT) SauveFen (TPCRT).

DerouleFenBas

Déclaration

```
procedure DerouleFenBas(XB, YB, XH, YH, Lignes : byte);
```

Objet

Déroule la fenêtre désignée du nombre de lignes spécifiées, vers le bas.

Commentaires

Utilisez cette routine à la place de *InsLine* lorsque vous ne voulez pas déplacer le curseur ou que vous déroulez une partie de l'écran différente de la fenêtre actuelle.

La fenêtre à dérouler est définie par *XB, YB, XH* et *YH* (les coordonnées écran sont en mode absolu et non en mode relatif à la fenêtre actuelle). *Lignes* est le nombre de lignes à dérouler (si *Lignes* = 0, la fenêtre toute entière est effacée).

Comme *InsLine, DerouleFenBas* remplit les lignes qu'elle crée avec des espaces avec l'attribut vidéo actuel (*TextAttr*).

Si *BiosScroll* vaut "False", une routine spéciale de déroulement est utilisée pour supprimer les scintillements avec les adaptateurs graphiques couleurs.

Exemple

```
var
   I : Word;
begin
   ClrScr;
   for I := 1 to ((HauteurEcran - 2) * LargeurEcran) do
      Write('*');
   CadreFen(1, 1, LargeurEcran, HauteurEcran - 2, TextAttr, TextAttr,
      ' Démonstration de DerouleFenBas ');
   Delay(2000);
   DerouleFenBas(2, 2, Pred(LargeurEcran), HauteurEcran - 3, 1);
end.
```

Remplit l'écran avec des lignes d'astérisques, dessine un cadre autour de la fenêtre puis déroule la fenêtre à l'intérieur du cadre d'une ligne vers le bas.

Voir aussi

DerouleFenHaut (TPCRT)

DerouleFenHaut

Déclaration

```
procedure DerouleFenHaut(XB, YB, XH, YH, Lignes : byte);
```

Objet

Déroule la fenêtre sélectionnée du nombre de lignes spécifié, vers le haut.

Commentaire

Utilisez cette routine à la place de *DelLine* lorsque vous ne voulez pas déplacer le curseur ou que vous déroulez une partie de l'écran différente de la fenêtre actuelle.

La fenêtre à dérouler est définie par *XB, YB, XH* et *YH* (ces coordonnées écran sont en mode absolu et non en mode relatif à la fenêtre actuelle).

Lignes est le nombre de lignes à dérouler. Si *Lignes* = 0, la fenêtre toute entière est effacée.

Comme *DelLine*, la routine *DerouleFenHaut* remplit les lignes qu'elle crée avec des espaces affichés avec l'attribut vidéo actuel (*TextAttr*).

Si *ScrollBios* vaut "False", une routine de déroulement spéciale est utilisée pour supprimer les parasites et les scintillements sur l'adaptateur couleur graphique.

Exemple

```
var
   I : Word;
begin
   ClrScr;
   for I := 1 to ((HauteurEcran - 2) * (LargeurEcran)) do
      Write('*');
   CadreFen(1, 1, LargeurEcran, HauteurEcran - 2, TextAttr, TextAttr,
      ' Démonstration de DerouleFenHaut ');
   Delay(2000);
   DerouleFenHaut(2, 2, Pred(LargeurEcran), HauteurEcran - 3, 1);
end.
```

Remplit les lignes de l'écran avec des astérisques, dessine un cadre autour de la fenêtre puis déroule la fenêtre dans le cadre d'une ligne vers le haut.

Voir aussi

DerouleFenBas (TPCRT)

EcritAttr

Déclaration

```
procedure EcritAttr(Ch : string; Ligne, Col : Byte);
```

Objet

Ecrit la chaîne d'attribut *Ch* en *Ligne, Col* sans changer les caractères.

Commentaires

Cette routine prend une chaîne d'attributs vidéo et les écrits sur l'écran en Ligne, Col. Les caractères qui se trouvent sur l'écran ne sont pas perturbés. *EcritAttr* est utilisée avec *LitAttr*.

Exemple

```
LitAttr(LargeurEcran, 1, 1, S);
for I := 1 to LargeurEcran do
   S[I] := Swap(S[I]) and $7F;
EcritAttr(S, 1, 1);
```

Inverse toutes les couleurs du fond et du premier plan de la ligne 1.

Voir aussi

LitAttr (TPCRT) EcritAttrFen (TPCRT)

EcritAttrFen

Déclaration

```
procedure EcritAttrFen(Ch : string; Ligne, Col : byte);
```

Objet

Ecrit une chaîne d'attributs *Ch* dans la fenêtre en *Ligne, Col* sans changer les caractères.

Commentaires

Cette routine se superpose à la routine *EcritAttr*. Elle modifie les coordonnées écran relatives de la fenêtre en coordonnées écran absolues puis appelle *EcritAttr*.

Remarquez que la procédure *EcritAttrFen* n'effectue aucun contrôle. Cela suppose que les coordonnées de la fenêtre passées en paramètre sont correctes et que les attributs sont adaptés à la fenêtre actuelle.

Voir aussi

LitAttrFen (TPCRT) EcritAttr (TPCRT)

EcritFacile

Déclaration

```
procedure EcritFacile(Ch : string; Ligne, Col : Byte;
                      var AttrsF : AttrsFaciles);
```

Objet

Ecrit *Ch* en *Ligne, Col* avec possibilité de choix des couleurs.

Commentaires

Cette routine est semblable à *EcritRapide* mais le dernier paramètre est un tableau d'octets et non un simple octet indiquant l'attribut vidéo de la chaîne. L'objet du tableau est de permettre à différentes parties de la chaîne d'être affichées avec des attributs différents et le signe utilisé pour indiquer un changement des attributs est un caractère de contrôle : ^A, ^B, ^C.

L'attribut vidéo par défaut pour la chaîne est AttrsF[0]. Pour modifier les attributs, vous devez insérer un ^A (pour *AttrsF[1]*), ^B (pour *AttrsF[2]*), ^C (pour *AttrsF[3]*). Pour renvoyer l'attribut précédent, vous devez insérer le même caractère de contrôle une deuxième fois. Voir les exemples ci-dessous.

Notez qu'à chaque fois que vous changez l'attribut deux octets sont utilisés sur la pile (qu'il faut restituer lorsque vous restaurer l'attribut précédent ou lorsque la routine se termine quel que soit l'ordre de ces deux événements). C'est au programmeur de vérifier éventuellement qu'il y a suffisamment de place sur la pile.

Voir GESTEMPS.PAS et CALC.PAS pour d'autre exemples d'utilisation de *EcritFacile*.

Exemple

```
const AttrsF : AttrsFaciles = ($7, $F, $E, $70);
...
EcritFacile('Ceci '^A'est'^A' '^B'un'^B' '^C'test', 1, 1, AttrsF);
```

Dans cet exemple, chacun des quatre mots de la chaîne est affiché avec un attribut vidéo différent. Le premier mot et les espace entre les mots sont tous affichés avec l'attribut par défaut (*AttrsF[0]*).

```
EcritFacile('Ceci '^A'est aussi un '^B'test', 2, 1, AttrsF);
```

Le premier mot est affiché avec l'attribut *AttrsF[0]*, les trois suivants avec *AttrsF[1]* et le dernier mot avec l'attribut *AttrsF[2]*.

Voir aussi

EcritRapide (TPCRT)

EcritFacileFen (TPCRT)

EcritFacileFen

Déclaration

```
procedure EcritFacileFen(Ch : string ; Ligne, Col : Byte ;
                         var AttrsF : AttrsFaciles) ;
```

Objet

Ecrit une chaîne dans la fenêtre en *Ligne, Col* avec possibilité de choisir la couleur.

Commentaires

Cette routine se superpose à la routine *EcritFacile*. Elle modifie les coordonnées écran relatives à la fenêtre en coordonnées écran absolues puis appelle *EcritFacile* pour afficher la chaîne.

Remarquez qu'aucun contrôle n'est effectué par *EcritFacileFen* ; cela suppose que les cordonnées passées en paramètre sont valides et que la chaîne se cadre exactement dans la fenêtre.

Voir aussi

EcritFacile (TPCRT)

EcritFenCompactee

Déclaration

```
procedure EcritFenCompactee(PFC : PtrFenCompactee; NomF : string);
```

Objet

Range la fenêtre compactée pointée par *PFC* dans *NomF*.

Commentaires

Cette routine vous permet d'enregistrer une fenêtre compactée créée par *CompacteFen* comme un fichier binaire. Si une erreur survient, le code d'erreur est renvoyé dans *ErreurCrt*. Outre les codes d'erreur d'entrée/sortie habituels, la seule valeur possible pour *ErreurCrt* est *CrtPointeurNil* (qui signifie que *PFC* vaut nil). N'oubliez pas que cette routine détruira tout fichier de nom *NomF* sans vous demander de confirmation.

Exemple

```
PFC := CompacteFen(1, 1, 80, 25);
EcritFenCompactee(PFC, 'Fenetre.dat');
```

Sauve tout l'écran comme une fenêtre compactée et le range dans FENETRE.DAT.

Voir aussi

CompacteFen (TPCRT) EcritFenCompactee (TPCRT)

EcritVite

Déclaration

```
procedure EcritVite(St : String; Ligne, Col, Attr:  byte);
```

Objet

Ecrit la chaîne *St* en *Ligne, Col,* avec *Attr* (attribut vidéo) sans scintillement.

Commentaire

Cette routine permet d'afficher un texte à l'écran à vitesse rapide sans scintillement. Contrairement à l'instruction standard *Write, EcritVite* vous donne le contrôle total de l'endroit où la chaîne est affichée et avec quel attribut vidéo, tout en éliminant le recours à une séquence lourde telle que :

```
GotoXY(X,Y);
TextColor(N);
TextBackground(N);
Write(S);
```

La seule restriction de *EcritVite* est qu'elle doit être utilisée seulement en mode texte. *St* est la chaîne affichée, *Ligne* et *Col* spécifient l'endroit où la chaîne doit être affichée (remarquez que ces coordonnées écran sont en mode absolu et non en mode relatif à la fenêtre actuelle). *Attr* est son attribut vidéo (voir la documentation sur REF).

Exemple

```
var
   Ligne, I : Byte;
begin
   ClrScr;
   Write('Pressez une touche pour une démonstration d''EcritVite...');
   if LitMotTouche = 0 then {ne rien faire};
   while not KeyPressed do
      for Ligne := 1 to HauteurEcran do
         for I := 0 to 7 do
            EcritVite('  Bonjour   ', Ligne, Succ(I * 10),
                      Lo(Random(127)));
   if LitMotTouche = 0 then {ne rien faire};
   ClrScr;
end.
```

Voir aussi

EcritMenVite (TPECRAN) EcritViteFen (TPCRT)

EcritViteFen

Déclaration

```
procedure EcritViteFen(Ch : string; Ligne, Col, Attr : Byte);
```

Objet

Ecrit une chaîne en utilisant les coordonnées relatives de fenêtre.

Commentaires

Cette routine se superpose à la routine *EcritVite*. Elle modifie les coordonnées écran relatives de la fenêtre en coordonnées écran absolues puis appelle *EcritVite* pour afficher la chaîne. Si la variable globale *FixeCouleurs* vaut "True", l'attribut vidéo Attr est adapté avant d'appeler *EcritVite*.

Remarquez qu'aucun contrôle n'est effectué par *EcritViteFen*. Cela suppose que les coordonnées de la fenêtre passée en paramètre sont correctes et que la chaîne se cadre exactement dans la fenêtre.

Dans les versions précédentes de *Turbo Professionnel*, cette routine se trouvait dans TPFENETR.

Exemple

```
Window(7, 5, 73, 20);
EcritViteFen('Bonjour', 1, 1, $E);
```

Affichera le mot 'Bonjour' en ligne 5, colonne 7. Si *FixeCouleurs* vaut "True" et *ModeActuel* est égal à mono, l'attribut vidéo sera converti en $F (blanc / noir) ; sinon l'attribut vidéo spécifié (jaune / noir) sera utilisé.

Voir aussi

EcritVite (TPCRT)
FixeCouleur (TPCRT)
EcritViteEtCoupe (TPFENETR)

EnregistreCoordsFen

Déclaration

```
procedure EnregistreCoordsFen(var CF : CoordFen);
```

Objet

Enregistre les coordonnées de la fenêtre active.

Commentaires

Cette routine vous permet de sauvegarder les coordonnées de la fenêtre active. Ces coordonnées peuvent être restaurées par la suite grâce à la procédure *RestaureCoordsFen*.

Exemple

```
var  CF : CoordFen;
begin
   EnregistreCoordsFen(CF);
   Window(1, 1, 80, 10);
   ...
   RestaureCoordsfen(CF);
end;
```

Voir aussi

RestaureCoordsFen(TPCRT) EnregistreCoordsFen (TPECRAN)

FermeBiblio

Déclaration

```
procedure FermeBiblio(var f: file; var PR : PtrRepertoire);
```

Objet

Ferme la bibliothèque *F* et libère son répertoire.

Commentaires

Cette routine ferme un fichier bibliothèque (*F*) précédemment ouvert par *OuvreBiblio* et libère la mémoire allouée pour le répertoire du fichier (pointé par PR). Le code d'erreur d'entrées/sorties pouvant survenir lors de la fermeture du fichier est renvoyé dans *ErreurCrt*. A la sortie, *PR* vaut nil.

Exemple

```
PR := OuvreBiblio(MaBib, 'MABIB.BIB');
...
FermeBiblio(MaBib, PR);
```

Ouvre la bibliothèque MABIB.BIB, la ferme et libère le répertoire.

Voir aussi

OuvreBiblio (TPCRT)

FixeCouleur

Déclaration

```
function FixeCouleur(C : byte):Byte;
```

Objet

Définit un attribut d'affichage en mono/noir et blanc.

Commentaire

Si la constante globale typée *FixeCouleurs* vaut "False", cette fonction renvoie simplement le paramètre d'entrée inchangé, sinon si le mode vidéo actuel (*ModeActuel*) est monochrome ou noir et blanc mixte, *FixeCouleur* adaptera l'attribut d'entrée pour un affichage correct sans couleur. Un résultat parfait est impossible mais cette fonction donne d'assez bons résultats.

Le paramètre C spécifie une combinaison de couleurs de fond et de premier plan (clignotement et couleur de fond dans les quatre bits de poids fort, couleur de premier plan dans les quatre bits de poids faible). Le paramètre passé doit contenir l'attribut d'affichage couleur. *FixeCouleur* renvoie une valeur adaptée à un affichage monochrome.

Les routines *InitFen* de TPFENETR appellent *FixeCouleur* comme la routine *SousMenu* dans TPMENU et la plupart des routines d'écriture de TPCRT (*EcritFacileFen* est une exception notable).

Votre programme peut l'appeler toutes les fois que des attributs vidéo sont utilisés.

Exemple

```
EcritVite('Bonjour', 1, 1, FixeCouleur($1C));
```

Avec un système couleur, affichera le message "Bonjour" en rouge sur fond bleu. Si *FixeCouleurs* vaut "True" et que le mode vidéo actuel vaut 0, 2 ou 7, la chaîne aura la couleur blanc clair sur fond noir.

Essayez d'exécuter le programme de démonstration DEROULFE en couleurs et en noir et blanc pour découvrir le fonctionnement de *FixeCouleur*.

Voir aussi

Déclarations (TPCRT).

Fonte8x8Sel

Déclaration

```
function Fonte8x8Sel : boolean;
```

Objet

Retourne "True" si le mode EGA ou VGA est actif et que la police 8x8 est utilisée.

Commentaire

Cette routine retournera "True" si un adaptateur EGA est actuellement actif et en mode 43 lignes ou si un adaptateur VGA est installé et en mode 50 lignes. L'appel de *Fonte8x8Sel* a aussi pour effet de réinitialiser la variable globale *HauteurActuelle*.

Remarque : TPCRT reconnait automatiquement les modes 43 lignes/50 lignes et initialise les coordonnées de fenêtres par défaut à 1,1,80,43/50. Si votre application fonctionne avec un affichage de 25 lignes, vous devrez ajouter le code suivant au début de votre programme :

```
Window(1, 1, 80, 25); {fenêtre de 25 lignes}
if Fonte8x8Sel then
   SelFonte8x8(False);
```

Exemple

```
if AffichageActuel >= EGA then
   if Fonte8x8Sel then
      SelFonte8x8(False)
   else
      SelFonte8x8(True);
```

Change un mode EGA de 25 lignes en un mode EGA à 43 lignes (ou un mode VGA de 25 à 50 lignes) et réciproquement.

Voir aussi

SelFonte8x8 (TPCRT)

HercPresent

Déclaration

```
function HercPresent : boolean;
```

Objet

Renvoie "True" si la carte graphique Hercules est présente.

Commentaires

Cette routine renvoie "True" si la carte graphique Hercules est installée dans le système. Cela ne signifie pas nécessairement qu'une carte Hercules soit installée.

Exemple

```
if HercPresent then begin
   Writeln('Ce programme ne peut pas fonctionner avec une
            carte Hercules');
   Halt;
end;
```

Arrête le programme si un adaptateur vidéo Hercules est installé.

Voir aussi

ModeGraphHerc (TPCRT)

TesteModeHerc (TPCRT)

IndicsClavier

Déclaration

```
function IndicsClavier : byte;
```

Objet

Renvoie les indicateurs d'état du clavier fournis par le BIOS.

Commentaire

Cette routine renvoie la valeur actuelle des indicateurs d'état du clavier fournis par le BIOS, c'est-à-dire la valeur binaire de l'octet rangé en $40:$17. Bien que cette valeur puisse être obtenue directement en utilisant une variable absolue, la fonction *IndicsClavier* utilise l'interruption $16 du BIOS et la fonction numéro 2. La signification de chaque bit de l'indicateur est donnée ci-dessous.

Bit	Valeur	Signification
7	$80	Touche (INS) active
6	$40	Touche (CAPS LOCK) active
5	$20	Touche (NUM LOCK) active
4	$10	Touche (SCROLL LOCK) active
3	$08	Touche (ALT) pressée
2	$04	Touche (CTRL) pressée
1	$02	Touche (SHIFT) gauche pressée
0	$01	Touche (SHIFT) droite pressée

Exemple

```
var
   OctetTestClavier : Byte;
begin
   OctetTestClavier := IndicsClavier;
   if (OctetTestClavier and $10) <> 0 then
      Writeln('Touche Scroll Lock enfoncée');
end.
```

InitBordCrt

Déclaration

```
procedure InitBordCrt(Attr : byte);
```

Objet

Positionne la bordure de l'écran avec l'attribut *Attr* si le mode et le type de la carte le permettent.

Commentaires

InitBordCrt vous permet de positionner la couleur de la bordure de l'écran avec l'attribut vidéo spécifié (Attr) sur les adaptateurs couleurs (CGA, MCGA, EGA, VGA). La procédure est inopérante si l'affichage courant correspond à une carte monochrome. Ne vous attendez pas à des résultats très satisfaisants avec des cartes EGA.

Exemple

```
TextColor(White);
TextBackground(Blue);
ClrScr;
InitBordCrt(Blue);
```

Initialise les couleurs de l'écran à blanc sur fond bleu, et la bordure de l'écran à bleu.

InitCaracsCadre

Déclaration

```
procedure InitCaracsCadre(Vertical, Horizontal, BasD, HautD, BasG,
                          HautG : char);
```

Objet

Initialise les caractères de cadre utilisés dans les appels à *CadreFen*.

Commentaire

Cette routine permet de modifier les caractères utilisés par *CadreFen* pour dessiner un cadre autour d'une fenêtre. Si vous le souhaitez, vous pouvez également modifier un ou plusieurs de ces caractères directement en changeant la variable globale *CaracsCadre*.

Exemple

```
InitCaracsCadre(#179, #196, #217, #191, #192, #218);
CadreFen(1, 1, LargeurEcran, HauteurEcran, TextAttr, TextAttr,
   ' L''entête se place ici ');
```

Dessine un cadre composé d'une seule ligne autour de l'écran.

Voir aussi

CadreFen (TPCRT)

InitClign

Déclaration

```
procedure InitClign(On : boolean);
```

Objet

Active l'attribut de clignotement en mode texte si *On* vaut "True".

Commentaires

InitClign active ou désactive le clignotement d'un texte. Si le clignotement est désactivé, les couleurs "lumineuses" telles que le rose ($D) et le jaune ($E) peuvent être utilisées comme couleurs d'arrière-plan.

Pour observer les résultats de *InitClign* ("False"), chargez le programme de démonstration REF. Pressez (CTRL)-(SHIFT)-R (shift droite) pour le lancer, (CTRL)-(PGDN) pour aller en fin de table, puis (ALT)-B pour désactiver le clignotement. Toutes les valeur supérieures à 127 correspondent à des couleurs avec un fond plus lumineux.

Exemple

```
InitClign(True);
TextAttr := $8F;
WriteLn('Le texte clignote');
InitClign(False);
WriteLn('Le texte ne clignote pas, même si l''attribut n''a pas
        changé');
```

Le première instruction est affichée en blanc sur un fond noir clignotant. La seconde est affichée en blanc sur un fond gris foncé.

InitModeHerc

Déclaration

```
procedure InitModeHerc(ModeGraph : boolean; PageGraphique : Byte);
```

Objet

Positionne la carte Hercules en mode graphique ou en mode texte.

Commentaires

Cette routine vous permet de basculer une carte Hercules du mode graphique (*ModeGraph* = "True") au mode texte (*ModeGraph* = "False"). Si *ModeGraph* est à "True", *PageGraphique* spécifie quelle page graphique doit être la page active (0 ou 1). Le paramètre *PageGraphique* est ignoré si *ModeGraph* vaut "False".

Exemple

```
GraphHerc := ModeGraphHerc;
if GraphHerc then InitModeHerc(False, 0);
...
if GraphHerc then InitModeHerc(True, 0);
```

Bascule une carte graphique Hercules en mode texte puis revient en mode graphique.

Voir aussi

ModeGraphHerc (TPCRT)

HercPresent (TPCRT)

InitPageVisible

Déclaration

```
procedure InitPageVisible(NumPage : byte);
```

Objet

Initialise la page vidéo actuelle.

Commentaire

Cette routine permet d'initialiser la page vidéo actuelle sur un adaptateur couleur (CGA, EGA, MCGA ou VGA) et n'a aucun effet sur les adaptateurs monochromes. En mode texte 40 colonnes, il y a huit pages vidéo disponibles (numérotées de 0 à 7). En mode texte 80 colonnes, il y a quatre pages vidéo (numérotées de 0 à 3). Remarquez que la page vidéo actuelle est toujours enregistrée dans la variable globale *PageActuelle*.

Exemple

```
if ModeActuel = 7 then
   WriteLn('Carte couleur nécessaire')
else begin
   WriteLn('Voici la page vidéo 0');
   Delay(1000);
   InitPageVisible(1);
   ClrScr;
   WriteLn('Voici la page vidéo 1');
   Delay(1000);
   InitPageVisible(0);
end;
```

Voir aussi

InitPageActive (TPECRAN)

InitTailleCurs

Déclaration

```
procedure InitTailleCurs(LigneDebut, LigneFin: byte);
```

Objet

Initialise les lignes de début et de fin du curseur.

Commentaire

Comme beaucoup d'autres procédures de l'unité TPCRT (*CursBLoc*, *CursPlat*, *CursCache* et *CursNormal*), cette routine permet de modifier la taille du curseur. Contrairement aux autres, elle donne un contrôle complet sur les lignes de début et de fin. En mode texte monochrome (7), le curseur contient quatorze lignes de balayage numérotées du haut (début) vers le bas (fin) de 0 à 13. En mode texte couleur, le curseur contient huit lignes numérotées du haut (début) vers le bas (fin) de 0 à 7. Le curseur EGA comporte quatorze lignes de balayage, il émule normalement le curseur CGA (pour compatibilité avec les programmes existants) et peut être traité comme ayant seulement huit lignes de balayage dans les modes texte).

Exemple

```
if ModeActuel = 7 then
   LigneFin := $0D  {= 13}
else
   LigneFin := $07; {= 7}
InitTailleCurs(0, LigneFin);
```

Cette procédure fournit un curseur en forme de bloc (voir l'entrée de *CursBloc*).

Voir aussi

CursBloc (TPCRT)
CursPlat (TPCRT)
CursCache (TPCRT)
CursNormal (TPCRT).

LibereFenCompactee

Déclaration

```
procedure LibereFenCompactee(var PFC :  PtrFenCompactee);
```

Objet

Libère une fenêtre compactée et positionne *PFC* à nil.

Commentaires

Cette routine libère une fenêtre compactée qui a été créée avec *CompacteFen* ou la lit à partir d'un fichier en utilisant *LitFenCompactee* ou *LitFenBiblio*. A la sortie, *PFC* est positionné à nil.

Exemple

```
PFC := LitFenCompactee('Fenetre.dat');
AffFenCompactee(PFC);
LibereFenCompactee(PFC);
```

Lit une fenêtre compactée dans un fichier, l'affiche puis la libère.

Voir aussi

CompacteFen (TPCRT)
LitFenBiblio (TPCRT)
LitFenCompactee (TPCRT)

LigneDebutCurs

Déclaration

```
function LigneDebutCurs : Byte;
```

Objet

Renvoie la dernière ligne de balayage du curseur.

Commentaire

Cette routine renvoie la dernière ligne de balayage du curseur qui vaut soit 6 (en mode texte couleur) ou 12 (en mode texte monochrome). En mode EGA 43 lignes, la ligne de balayage de début est habituellement 5.

Exemple

```
CursNormal; {Sélectionne le curseur par défaut sur deux lignes}
WriteLn('Ligne de début pour le curseur : ', LigneDebutCurs);
```

Affichera 5, 6 ou 12 en fonction du mode actuel.

Voir aussi

LigneFinCurs (TPCRT)
CursDebFin (TPCRT)

LigneFinCurs

Déclaration

```
function LigneFinCurs : Byte;
```

Objet

Renvoie la dernière ligne du curseur.

Commentaires

Cette routine renvoie la dernière ligne de balayage du curseur qui est normalement la septième (en mode texte couleurs) ou 13 (en mode texte monochrome).

Exemple

```
CursNormal;
WriteLn('Ligne de fin pour le curseur = ', LigneFinCurs);
```

Affichera 7 ou 13 en fonction du mode vidéo actuel.

Voir aussi

LigneDebutCurs (TPCRT)

CursDebFin (TPCRT)

LitAttr

Déclaration

```
procedure LitAttr(Nombre, Ligne, Col : byte; var Ch : string);
```

Objet

Lit *Nombre* attributs sur l'écran dans *Ch* à partir de *Ligne, Col.*

Commentaires

Cette routine est semblable à la routine *LitVite* mais au lieu d'extraire les caractères de l'écran, elle extrait les attributs vidéo. Elle ressemble également à *LitAttrCurs* mais elle est plus rapide.

Nombre est le nombre d'attributs vidéo extraits de l'écran. *Ligne* et *Col* spécifient l'emplacement où commence la lecture et *Ch* est la variable chaîne dans laquelle les attributs extraits de l'écran seront renvoyés.

Exemple

```
LitAttr(LargeurEcran, 1, 1, S);
for I := 1 to LargeurEcran do
   S[I] := Swap(S[I]) and $7F;
EcritAttr(S, 1, 1);
```

Inverse toutes les couleurs de premier plan et de l'arrière plan sur la ligne 1

Voir aussi

LitVite (TPCRT)
LitAttrCurs (TPCRT)
EcritAttr (TPCRT)

LitAttrCurs

Déclaration

```
function LitAttrCurs : byte;
```

Objet

Renvoie les attributs de la position courante du curseur sur la page sélectionnée.

Commentaires

Cette routine est semblable à *LitCaracSousCurs* mais au lieu de renvoyer le caractère sous le curseur, elle renvoie l'attribut vidéo de ce caractère.

Exemple

```
CarSousCurseur := LitCaracSousCurs;
AttrSousCurseur := LitAttrCurs;
```

Range le caractère et l'attribut sous le curseur dans *CarSousCurs* et *AttrSousCurs*, respectivement.

Voir aussi

LitAttr (TPCRT) LitCaracSousCurs (TPCRT)

LitAttrFen

Déclaration

```
procedure LitAttrFen(Nombre, Ligne, Col : Byte; var Ch : string);
```

Objet

Lit *Nombre* attributs sur l'écran dans *Ch* à partir *Ligne, Col* de la fenêtre.

Commentaires

Cette routine se superpose à la routine *LitAttr*. Elle modifie les coordonnées écran relatives de la fenêtre en coordonnées écran absolues, puis appelle *LitAttr*.

Remarquez qu'aucun contrôle n'est effectué par *LitAttrFen*. Cela suppose que les coordonnées de la fenêtre passés en paramètres sont correctes et que les attributs à extraire de l'écran appartiennent à la fenêtre actuelle.

Voir aussi

EcritAttr (TPCRT)

LitCaracSousCurs

Déclaration

```
function LitCaracSousCurs : char;
```

Objet

Renvoie le caractère sous la position actuelle du curseur.

Commentaire

Comme *LitVite*, *LitCaracSousCurs* peut être utilisée pour lire des données sur l'écran. Bien que *LitCaracSousCurs* lise un seul caractère à la fois et exige que le curseur soit positionné correctement, elle possède l'avantage de fonctionner également dans les modes graphiques standard CGA. (En supposant que les instructions *Write* et *WriteLn* aient été utilisées pour afficher du texte). *LitVite* peut être utilisée seulement en mode texte.

Exemple

```
var
   X, Y : Byte;
   XY : Word;
begin
   XY := CoordXY;
   for Y := 1 to HauteurEcran do begin
      for X := 1 to LargeurEcran do begin
         GotoXY(X, Y);
         Write(Lst, LitCaracSousCurs);
      end;
      WriteLn(Lst);
   end;
   PlaceEnXYAbs(Lo(XY), Hi(XY));
end.
```

Cette routine donne le même résultat que la routine d'impression d'écran du BIOS.

Voir aussi

LitVite (TPCRT)

LitEtatCurs

Déclaration

```
procedure LitEtatCurs(Var XY, lignesBalayees : Word);
```

Objet

Renvoie la position actuelle et la taille du curseur.

Commentaire

Cette routine, lorsqu'elle est utilisée avec *RestaureEtatCurs,* fournit un moyen pratique de sauvegarder et de restaurer la taille et la position actuelles du curseur. Elle est utile non seulement dans les routines "popup" écrites avec TPTSR mais aussi dans les boîtes noires (par exemple, *LitChaine* de TPEDIT) qui doivent pouvoir manipuler la taille du curseur sans affecter l'affichage du programme appelant.

Exemple

```
var
   XY, LigBalayeesParCurseur, Tche : Word;
begin
   LitEtatCurs(XY, LigBalayeesParCurseur);
   CursCache;
   EcritVite('Erreur. Pressez une touche pour continuer...', 25, 1,
         TextAttr);
   Tche := LitMotTouche;
   RestaureEtatCurs(XY, LigBalayeesParCurseur);
end.
```

Voir aussi

RestaureEtatCurs (TPCRT)

LitFenBiblio

Déclaration

```
function LitFenBiblio(var F : File; Pr : PtrRepertoire;
                      NomFen : NomBiblio): PtrFenCompactee;
```

Objet

Lit une fenêtre compactée dans une bibliothèque et renvoie un pointeur sur cette fenêtre.

Commentaires

Cette routine charge en mémoire une fenêtre compactée qui a été sauvegardée dans une bibliothèque avec *AjoutFenBiblio*. *F* est le fichier bibliothèque, *Pr* est le pointeur sur son répertoire et *NomFen* est le nom de la fenêtre qui doit être lue en mémoire.

En cas de succès, un pointeur sur la fenêtre compactée est renvoyé. En cas d'erreur, elle renvoie nil et *ErreurCrt* contient le code d'erreur. Outre le codes d'erreurs d'entrée/sortie habituels, les seules valeurs possibles pour *ErreurCrt* sont *CrtMemInsuf* (pas assez de mémoire pour allouer la fenêtre compactée) et *CrtFenIncorrecte* (le fichier spécifié par *F* ne contient pas une fenêtre compactée correcte).

Exemple

```
PR := OuvreBiblio(MaBib, 'MABIB.BIB');
PFC := LitFenBiblio(MaBib, PR, 'MAFEN.1');
AffFenCompactee(PFC);
LibereFenCompactee(PFC);
FermeBiblio(MaBib, PR);
```

Lit une fenêtre compactée dans la bibliothèque, l'affiche puis, la libère et ferme la bibliothèque.

Voir aussi

AjoutFenBiblio (TPCRT)
CreeBiblio (TPCRT)
OuvreBiblio (TPCRT)
FermeBiblio (TPCRT)

LitFenCompactee

Déclaration

```
function LitFenCompactee(NomF : string): PtrFenCompactee;
```

Objet

Lit la fenêtre compactée enregistrée dans *NomF* en mémoire et renvoie un pointeur sur celle-ci.

Commentaires

Cette routine lit en mémoire une fenêtre compactée sauvegardée sur disque (sous le nom *NomF*) par *EcritFenCompactee*. En cas de succès, elle renvoie un pointeur sur la fenêtre compactée. Si une erreur se produit, elle renvoie nil et *ErreurCrt* contient un code erreur. Outre les erreurs d'entrée/sortie habituelles, les valeurs possibles de *ErreurCrt* sont *CrtFenIncorrecte* (le fichier nommé par *NomF* ne contient pas une fenêtre compactée correcte) et *CrtMemInsuf* (pas assez de mémoire pour allouer la fenêtre compactée).

Exemple

```
PFC := LitFenCompactee('Fenetre.dat');
AffFenCompactee(PFC);
LibereFenCompactee(PFC);
```

Lit une fenêtre compactée sur fichier, l'affiche et la libère.

Voir aussi

CompacteFen (TPCRT)

EcritFenCompactee (TPCRT)

LitModeVideo

Déclaration

```
Function LitModeVideo : byte;
```

Objet

Lit le mode vidéo actuel.

Commentaire

LitModeVideo réinitialise également les variables globales de touches utilisées par TPCRT : *ModeActuel* (mode vidéo actuel), *EnModeTexte* (faux en mode graphique), *LargeurEcran* (largeur actuelle de l'écran), *PageActuelle* (page actuelle affichée) et *SegmentVideo* (segment de l'adresse de début de la mémoire vidéo). Elle est principalement utilisée avec les routines "popup" à la place de la routine *ReinitVideo* qui réinitialise plusieurs variables globales et prend donc plus de temps.

Un bon exemple d'utilisation de *LitModeVideo* se trouve dans le programme GESTEMPS.PAS. GESTEMPS appelle cette routine avant d'afficher son horloge écran pour s'assurer que l'horloge est affichée dans le coin supérieur droit de l'écran, que l'affichage n'est pas en mode graphique et que l'horloge est affichée dans la page vidéo actuelle de l'adaptateur actif.

Voir aussi

Fonte8x8Sel (TPCRT)

ReinitVideo (TPCRT)

LitMotTouche

Déclaration

```
function LitMotTouche : Word;
```

Objet

Attend la frappe d'une touche puis retourne le code de recherche et le code du caractère.

Commentaire

Contrairement à *ReadKey*, cette routine renvoie à la fois le code de balayage et le code du caractère pour la touche pressée. L'octet de poids faible du mot retourné contient le code du caractère et l'octet de poids fort contient le code de balayage.

LitMotTouche a deux autres avantages sur *ReadKey* ; le premier est de vous permettre d'obtenir toutes les données dont vous avez besoin avec un seul appel de fonction, le deuxième est de vous permettre de faire la différence entre les touches qui partagent le même code de caractère mais qui possèdent des codes de balayage différents. Par exemple, sur un clavier de PC ou d'AT, il y a deux touches qui renvoient '+' (#43) : l'une dans la ligne supérieure du clavier principal et l'autre sur le pavé numérique. La première a un code de balayage égal à 13 ($2B) et la seconde a un code de balayage de 78 ($4E). En utilisant *LitMotTouche*, vous pourrez aisément distinguer ces deux codes, comme dans l'exemple ci-dessous.

Exemple

```
var
   Tche : Word;
   Ch   : Char absolute Tche;
begin
   Tche := LitMotTouche;
   case Ch of
      '+' : if (Hi(Tche) = $4E) then
                WriteLn('Vous avez pressé la touche "+" grise')
            else
                WriteLn('Vous avez pressé la touche "+" blanche')
   end;
end.
```

Voir aussi

ContrlClavier (CPCRT)

LitVite

Déclaration

```
procedure LitVite(Nombre, Ligne, Col : byte; var St : string);
```

Objet

Lit un nombre de caractères à l'écran dans *St* en commençant en *Ligne, Col.*

Commentaires

Comme *LitCaracSousCurs, LitVite* lit un texte sur l'écran mais beaucoup plus rapidement. Elle est le plus souvent utilisée dans les programmes écrits avec l'unité TPTSR qui peuvent explorer le contenu de l'écran affiché par le programme interrompu.

Nombre est le nombre de caractères à lire sur l'écran. *Ligne* et *Col* indiquent l'emplacement où doit commencer la lecture (remarquez que ces coordonnées écran sont en mode absolu et non en mode relatif dans la fenêtre actuelle). *St* est une chaîne de caractères dans laquelle les caractères lus sur l'écran sont renvoyés.

Voir aussi

LitLigneVite (TPECRAN) LitCaracSousCurs (TPCRT)

LitViteFen

Déclaration

```
procedure LitViteFen(Nombre, Ligne, Col : byte; var Ch : string);
```

Objet

Lit *Nombre* caractères de l'écran dans *Ch* à partir *Ligne, Col* de la fenêtre.

Commentaires

Cette routine se superpose à la routine *LitVite*. Elle transforme les coordonnées écrans relatives à la fenêtre en coordonnées écrans absolues, puis appelle *LitVite* pour lire la chaîne sur l'écran.

Remarquez qu'aucune vérification n'est faite par *LitViteFen*. Cela implique que les coordonnées de la fenêtre passée en paramètre sont correctes et que la chaîne lue sur l'écran est bien cadrée dans la fenêtre actuelle.

Exemple

```
EcritViteFen('Bonjour', 1, 1, 7);
LitViteFen(5, 1, 1, Chne);
```

Lira la chaîne 'Bonjour' dans *Chne.*

ModeGraphHerc

Déclaration

```
function ModeGraphHerc : Boolean ;
```

Objet

Renvoie "True" si la carte Hercules est en mode graphique.

Commentaires

Cette routine renvoie "True" si la carte Hercules (ou toute carte compatible), est installée en mode graphique. Prenez conscience que le test utilisé ici est loin d'être parfait. Il fonctionne correctement avec de vraie carte Hercules et il fonctionne plus ou moins bien avec des cartes compatibles Hercules mais dans de nombreux cas il peut être défaillant. Les cas les plus fréquents sont les cartes compatibles EGA/VGA qui fournissent une émulation Hercules. Pour déterminer si la routine est ou n'est pas fiable, appelez *TesteModeHerc*. Remarquez qu'il n'est pas possible de déterminer quelle page graphique (0 ou 1) est active.

Exemple

```
if HercPresent then begin
   if TesteModeHerc and ModeGraphHerc then
   InitModeHerc(False, 0);
end;
```

Vérifie si une carte Hercules est installée en mode graphique. Si c'est le cas, l'affichage est basculé en mode texte.

Voir aussi

HercPresent (TPCRT)

InitModeHerc (TPCRT)

OuvreBiblio

Déclaration

```
function OuvreBiblio(var F : File; Nom : string): PtrRepertoire;
```

Objet

Ouvre la bibliothèque spécifiée et renvoie un pointeur sur son répertoire.

Commentaires

Cette routine ouvre une bibliothèque crée précédemment par *CreeBiblio* et en cas de succès, renvoie un pointeur sur son répertoire. Si une erreur apparaît, la fonction renvoie nil et *ErreurCrt* contient le code erreur. Outre les codes d'erreur d'entrée/sortie habituels, les seules valeurs possible de *ErreurCrt* sont *CrtBiblioNonValide* (le fichier est incorrect) ou *CrtMemInsuf* (pas assez de mémoire pour allouer le répertoire).

Exemple

```
PFC := CompacteFen(1, 1, 80, 25);
PR := OuvreBiblio(MaBib, 'MABIB.BIB');
AjoutFenBiblio(PFC, MaBib, PR, 'MAFEN.1');
```

Compacte tout l'écran puis le sauve sous le nom MAFEN.1 dans MABIB.BIB.

Voir aussi

CreeBiblio (TPCRT)

PlaceChaine

Déclaration

```
procedure PlaceChaine(C : string);
```

Objet

Place le contenu de C dans le buffer du clavier.

Commentaires

Cette routine insère la chaîne spécifiée directement dans le buffer du clavier. La chaîne doit contenir au maximum 16 caractères (en supposant que le buffer du clavier soit vide). Les caractères supplémentaires sont ignorés. Avec *PlaceTouche,* cette routine peut être utilisée pour mettre en oeuvre un système de macros clavier simples aussi bien que pour passer des touches à des programmes exécutés avec *Exec* (Dos) ou *ExecDos* (TPDOS).

Exemple

```
PlaceChaine('Bonjour à tous '^M);
ReadLn(MaChaine);
```

Chargera ma chaîne avec 'Bonjour à tous !'.

PlaceEnXYAbs

Déclaration

```
procedure PlaceEnXYAbs(X,Y : byte);
```

Objet

Place le curseur en colonne *X*, ligne *Y*.

Commentaire

Comme *GotoXY*, *PlaceEnXYAbs* déplace le curseur à une adresse spécifique de l'écran mais les coordonnées passées sont en mode absolu et non en mode relatif. Toutes les coordonnées ont pour origine le coin supérieur gauche (1,1) et aucun contrôle d'intervalle n'est effectué (le curseur peut être déplacé en dehors de la fenêtre actuelle).

Exemple

```
Window(3, 3, 80, 25);
PlaceEnXYAbs(3, 3);
```

Place le curseur dans le coin supérieur gauche de la fenêtre comme *'GotoXY (1,1)'*.

Voir aussi

CoordXAbs (TPCRT)
CoordYAbs (TPCRT)
CoordXY (TPCRT)

PlaceTouche

Déclaration

```
procedure PlaceTouche(W : Word);
```

Objet

Place une touche dans le buffer du clavier.

Commentaires

Cette routine insère le code de la touche spécifiée directement dans le buffer du clavier. Si le buffer du clavier est plein, elle est inopérante.

Exemple

```
PlaceTouche($3B00);
```

Insère le code touche pour (F1) dans le buffer du clavier.

```
PlaceTouche(Ord('A'));
```

Insère le code touche de "A" dans le buffer du clavier (la plupart des programmes ignorent l'octet de poids fort du code touche si l'octet de poids faible est différent de zéro).

ReinitVideo

Déclaration

```
procedure ReinitVideo;
```

Objet

Réinitialise les variables internes de TPCRT.

Commentaire

ReinitVideo est conçue principalement pour les programmes résidents qui utilisent *ExecDos* ou *Exec* et les programmes qui utilisent les options graphiques. Elle réinitialise toutes les variables globales dépendantes du mode utilisées par TPCRT : *ModeActuel* (Mode vidéo actuel), *EnModeTexte* ("False" en mode graphique), *LargeurEcran* (largeur actuelle de l'écran), *HauteurEcran* (hauteur actuelle de l'écran), *WindMin* et *WindMax* (dimensions de la fenêtre, la fenêtre actuelle est égale à l'écran tout entier), *PageActuelle* (page vidéo actuelle), *AffichageActuel* (le type de l'adaptateur d'affichage actuel), *CheckSnow* (supprime les parasites sur les affichages couleurs) et *SegmentVideo* (valeurs du segment dans l'adresse de départ de la mémoire vidéo).

En général, il y a seulement trois situations dans lesquelles vous aurez besoin d'utiliser *ReinitVideo* : en entrée d'une unité empilée, après l'exécution d'un appel du DOS avec *ExecDos* (ou avec la routine *Exec* de Turbo) ou après être entré ou sorti du mode graphique.

Voir aussi

ExecDos (TPDOS)

LitModeVideo (TPCRT)

RemplitVite

Déclaration

```
procedure RemplitVite(Nombre : Word; Car: Char; Ligne, Col,
                      Attr: Byte);
```

Objet

Remplit *Nombre* mots vidéo avec *Car/Attr* à partir de *Ligne, Col.*

Commentaires

Comme la routine *RemplitCaracVideo* de TPECRAN, cette routine remplie une région contiguë de la mémoire vidéo en commençant en *Ligne, Col* avec *Nombre* occurrences du couple spécifié *Car/Attr* (*Car* et *Attr*). Un exemple d'utilisation serait de remplir une ligne entière de l'écran avec le caractère tiret (#196).

Exemple

```
RemplitVite(LargeurEcran, #196, 13, 1, $F);
```

Remplit la ligne 13 avec le caractère #196 affiché en blanc sur noir.

Voir aussi

RemplitCaracVideo (TPECRAN)

RemplitViteFen

Déclaration

```
procedure RemplitViteFen(Nombre : word; Car : Char;
                         Ligne, Col, Attr : Byte);
```

Objet

Remplit *Nombre* mots vidéo avec *Car/Attr* à partir de *Ligne, Col.*

Commentaires

Cette routine se superpose à la routine *RemplitVite*. Elle transforme les coordonnées relatives de la fenêtre spécifiées en coordonnées écran absolues puis appelle *RemplitVite* pour afficher le couple caractère/attribut.

Remarquer qu'aucun contrôle n'est effectué par *RemplitViteFen*. Cela suppose que les coordonnées de la fenêtre passée en paramètre sont correctes et que les caractères affichés, se cadreront bien dans la fenêtre actuelle.

RestaureCoordsFen

Déclaration

```
procedure RestaureCoordsFen(CF : CoordFen);
```

Objet

Restaure les coordonnées de la fenêtre précédemment sauvegardées.

Commentaires

Cette routine vous permet de restaurer les coordonnées de la fenêtre savegardées par la procédure *EnregistreCoordsFen*. Voir cette dernière routine pour un exemple.

Voir aussi

RestaureCoordsFen (TPECRAN) EnregistreCoordsFen (TPCRT)

RestaureEtatCurs

Déclaration

```
procedure RestaureEtatCurs(XY, lignesBalayees : Word);
```

Objet

Remet le curseur à la taille et la position enregistrées avec *LitEtatCurs*.

Commentaire

Cette routine, lorsqu'elle est utilisée avec *LitEtatCurs*, fournit un moyen pratique d'enregistrer et de restaurer plus tard la taille et la position actuelles du curseur. Elle est non seulement utile dans les routines "popup" écrites avec TPTSR mais aussi dans les "boîtes noires" (telle que *LitChaine* de TPEDIT) qui doivent manipuler la taille du curseur sans affecter l'affichage du programme appelant. Voir l'entrée de *LitEtatCurs* pour exemple.

Voir aussi

LitEtatCurs (TPCRT)

RestaureFen

Déclaration

```
procedure RestaureFen(XBas, YBas, XHaut, YHaut : byte;
                      Dealloue : boolean; var Couvre : Pointer);
```

Objet

Restaure le contenu de l'écran et libère l'espace du buffer si nécessaire.

Commentaire

Lorsqu'elle est utilisée avec *SauveFen*, cette routine fournit un moyen pratique de sauvegarder et de restaurer tout ou partie d'un écran (une fenêtre).

Les dimensions de la fenêtre à restaurer sont données par *XBas*, *YBas*, *XHaut*, et *YHaut*. (Remarquez que ces coordonnées écran sont en mode absolu et non en mode relatif dans la fenêtre actuelle). Bien que les coordonnées de fenêtre soient habituellement celles passées à *SauveFen*, elles ne sont pas forcément identiques. *Couvre* est un pointeur sur le buffer utilisé pour enregistrer les données de l'écran. *Dealloue* indique si le buffer sera libéré ou non une fois que la fenêtre sera restaurée).

Voir *SauveFen* pour un exemple et des explications.

Voir aussi

EffaceFen (TPFENETR)
SauveFen (TPCRT)
DeplaceEcran (TPCRT)

SauveFen

Déclaration

```
function SauveFen(XBas, YBas, XHaut, YHaut : byte; Alloue : boolean;
                  Var Couvre: Pointer) : boolean;
```

Objet

Alloue un buffer si nécessaire et sauvegarde le contenu de la fenêtre.

Commentaire

Utilisée avec *RestaureFen,* cette routine fournit un moyen pratique de sauvegarde et d'enregistrement de tout ou partie d'un écran (une fenêtre). Les dimensions de la fenêtre sauvegardée sont définies par *XBas, YBas, XHaut* et *YHaut*. (Remarquez que ces coordonnées écran sont en mode absolu et non en mode relatif dans la fenêtre actuelle). *Alloue* indique si un buffer doit être alloué ou non pour les données de l'écran. Si *Alloue* est à "True" un pointeur sur le buffer est retourné dans *Couvre*. Sinon *Couvre* doit être un pointeur dans un buffer suffisamment grand pour enregistrer les données. *SauveFen* renverra "False" seulement si *Alloue* est à "True" et s'il n'y a plus assez de place sur le tas pour allouer un buffer.

Puisque *SauveFen* utilise *DeplaceEcran* pour déplacer les données, les scintillements et les parasites sont supprimés.

Remarques : *SauveFen* et *RestaureFen* doivent être utilisées seulement en mode texte et non en mode graphique.

Exemple

```
var
   PtrBuf : Pointer;
begin
   if not SauveFen(1, 1, LargeurEcran, HauteurEcran, True, PtrBuf^)
   then
      Exit;
   {...Suite...}
   RestaureFen(1, 1, LargeurEcran, HauteurEcran, True, PtrBuf);
end.
```

Incorrect ! Le '^' dans PtrBuf est inutile.

```
   if not SauveFen(1, 1, LargeurEcran, HauteurEcran, True, PtrBuf)
then
```

Correct. Cette routine alloue un buffer et sauvegarde l'écran tout entier qui sera plus tard restauré avec *RestaureFen*.

Voir aussi

DeplaceEcran (TPCRT)

RestaureFen (TPCRT)

SelFonte8X8

Déclaration

```
procedure SelFonte8x8(On : boolean );
```

Objet

Active ou désactive la police 8x8 (mode 43 lignes ou 50 lignes).

Commentaire

Cette routine vous permet de sélectionner (*On* = "True") ou non (*On* = "False"), le mode EGA 43 lignes et le mode VGA 50 lignes. Si l'adaptateur EGA ou VGA n'est pas installé, elle ne fait rien. Après avoir basculé en mode 43 lignes/50 lignes, la variable globale *HauteurEcran* sera initialisée à 42 ou 49 (la hauteur actuelle moins un) plutôt qu'à sa valeur de visualisation égale à 24. Néanmoins, les coordonnées de la fenêtre actuelle ne seront pas modifiées car les effets ne sont pas toujours désirables. Leur adaptation aux nouvelles dimensions de l'écran est sous la responsabilité du programmeur.

Cette routine n'efface pas l'écran, seule la solution offerte par l'unité standard CRT le permet :

```
TextMode(ModeActuel + Font8X8);
```

Remarque : A cause d'une bizarrerie du BIOS avec EGA, le curseur normal de deux lignes (ligne de début = 6 et ligne de fin = 7) sera caché après avoir basculé en mode 43 lignes si l'émulation du curseur CGA est toujours en fonction. Pour éviter cela, *SelFonte8x8* désactive automatiquement l'émulation lors du passage au mode 43 lignes et l'active à nouveau lors du retour en mode 25 lignes.

Exemple

```
if not (AffichageActuel < Ega) then
   WriteLn('Carte EGA ou VGA nécessaire')
else begin
   Write('Voici le mode 25 lignes... ');
   Delay(1000);
   SelFonte8x8(True);
   WriteLn('et voici le mode ', HauteurEcran, ' lignes.');
   Delay(1000);
   SelFonte8x8(False);
end;
```

Voir aussi

Fonte8x8Sel (TPCRT)

SupprimeFenBiblio

Déclaration

```
procedure SupprimeFenBiblio(var F : file; PR : PtrRepertoire;
                            NomFen : NomBiblio);
```

Objet

Supprime une fenêtre compactée d'une bibliothèque donnée.

Commentaires

Cette routine supprime la fenêtre compactée désignée par *NomFen* de la bibliothèque donnée (F). *PR* pointe sur le répertoire de la bibliothèque. La bibliothèque doit avoir été précédemment ouverte par un appel à *CreeBiblio* ou *OuvreBiblio*. Remarquez que cette routine ne supprime pas physiquement la fenêtre de la bibliothèque. Elle la marque simplement comme étant supprimée logiquement du répertoire. Si vous voulez supprimer physiquement la fenêtre, appelez ensuite *CompacteBiblio*.

A la sortie, le code d'erreur éventuel est renvoyé dans *ErreurCrt*. Outre les codes d'erreurs d'entrées/sorties habituels, la seule valeur pour *ErreurCrt* est *CrtFenAbsente* (*NomFen* n'a pas été trouvée dans le répertoire de la bibliothèque).

Exemple

```
PR := OuvreBiblio(MaBib, 'MABIB.LBR');
SupprimeFenBiblio(MaBib, PR, 'MAFEN.1');
FermeBiblio(MaBib, PR);
CompacteBiblio('MABIB.LBR');
```

Marque MAFEN.1 comme étant supprimée logiquement de MABIBLIO.LBR puis compacte la bibliothèque pour supprimer physiquement cette fenêtre.

Voir aussi

CreeBiblio (TPCRT)
OuvreBiblio (TPCRT)
CompacteBiblio (TPCRT)

TesteModeHerc

Déclaration

```
function TesteModeHerc : boolean;
```

Objet

Renvoie "True" si *ModeGraphHerc* peut fonctionner.

Commentaires

Cette routine est prévue pour être utilisée en conjonction avec la routine *ModeGraphHerc*. Son objet est simplement de déterminer si le test utilisé par cette routine est fiable ou non.

Voir aussi

HercPresent (TPCRT) InitModeHerc(TPCRT)

TexteRapide

Déclaration

```
procedure TexteRapide(Ch : string; Ligne, Col : Byte);
```

Objet

Ecrit *Ch* en *Ligne, Col* sans modifier l'attribut vidéo.

Commentaires

Cette routine est semblable à *EcritRapide* mais ne comporte pas d'attribut vidéo. Le texte est écrit avec la couleur qui se trouve déjà à l'emplacement écran où la chaîne est écrite.

Exemple

```
TextAttr := $F;
ClrScr;
TexteRapide('Ce texte est affiché en blanc sur fond noir', 1, 1);
```

Affiche la chaîne donnée dans le coin supérieur gauche de l'écran avec la couleur déjà présente en ligne 1.

Voir aussi

EcritRapide (TPCRT)

TexteRapideFen

Déclaration

```
procedure TexteRapideFen(Ch : string; Ligne, Col : Byte);
```

Objet

Ecrit *Ch* dans la fenêtre en *Ligne, Col* sans modifier l'attribut vidéo.

Commentaires

Cette routine se superpose à la routine *TexteRapide*. Elle modifie les coordonnées écran relatives de la fenêtre en coordonnées absolues, puis appelle *TexteRapide* pour afficher la chaîne. Aucun contrôle n'est effectué par *TexteRapideFen*. Cela suppose que les coordonnées écran passées en paramètre soient correctes et que la chaîne se cadre exactement dans la fenêtre.

Exemple

```
Window(7, 5, 73, 20);
TexteRapideFen('Bonjour', 1, 1);
```

Affichera le mot 'Bonjour' en ligne 5, colonne 7.

VertVite

Déclaration

```
procedure VertVite(Ch : string; Ligne, Col, Attr : byte);
```

Objet

Ecrit *Ch* verticalement en *Ligne, Col* avec *Attr*

Commentaires

Cette routine est semblable à *EcritRapide*, mais elle écrit le texte verticalement et non horizontalement.

Exemple

```
EcritVite('Ecriture horizontale...', 1, 2, TextAttr);
VertVite('Ecriture verticale...', 2, 1, TextAttr);
```

La première chaîne sera écrite horizontalement dans le haut de l'écran et la seconde sera écrite verticalement le long du bord gauche de l'écran.

Voir aussi

EcritRapide (TPCRT)

VertViteFen

Déclaration

```
procedure VertViteFen(Ch : string; Ligne, Col : Byte);
```

Objet

Ecrit *Ch* verticalement dans la fenêtre en *Ligne, Col.*

Commentaires

Cette routine se superpose à la routine *VertVite*. Elle modifie les coordonnées écrans relatives de la fenêtre en coordonnées écrans absolues puis appelle *VertVite* pour afficher la chaîne. Si la variable globale *FixeCouleurs* vaut "True", l'attribut vidéo *Attr* est adapté avant d'appeler *VertVite*.

Remarquez qu'aucun contrôle n'est effectué par *VertViteFen*. Cela suppose que les coordonnées de la fenêtre passées en paramètre soient correctes et que la chaîne se cadre exactement dans la fenêtre.

Exemple

```
Window(7, 5, 73, 20);
VertViteFen('Bonjour', 1, 1, Attr);
```

Affichera le mot 'Bonjour' verticalement en colonne 7 en commençant à la ligne 5.

TPFENETR : Fenêtres "popup"

TPFENETR permet de gérer n'importe quel nombre de fenêtres texte affichées sur la page vidéo par défaut de l'écran physique. Elle permet d'implémenter des écrans virtuels de taille supérieure à l'écran physique et inférieure à 64K octets. Elle supporte des fenêtres qui se chevauchent ou des fenêtres disjointes comportant des cadres et des ombres, et elle comporte des options d'explosion de fenêtres.

Notez que TPFENETR et TPECRAN mettent en oeuvre des fonctions identiques mais différemment. Vous pouvez les utiliser séparément dans vos programmes mais pas conjointement. Chaque unité comporte une séquence d'initialisation qui vérifie si ces deux unités ne sont pas utilisées dans un même programme. Nous vous recommandons de choisir TPFENETR au lieu de TPECRAN si c'est possible. TPECRAN garantit la compatibilité avec les versions précédentes de *Turbo Professionnel.*

Une fenêtre est une zone rectangulaire de l'écran dont les caractéristiques sont décrites par un enregistrement fenêtre. Chaque enregistrement contient la position, la taille, les attributs vidéo, la forme et la position du curseur, le cadre et l'en-tête de la fenêtre ainsi que le buffer contenant la partie de l'écran recouverte par la fenêtre et (facultativement) la fenêtre elle-même.

Néanmoins, par souci méthodologique, TPFENETR cache ces détails de bas niveau et met à votre disposition des fonctions de haut niveau pour gérer les fenêtres plus facilement. TPFENETR contient des routines pour initialiser et libérer les enregistrements, pour afficher et effacer les fenêtres et pour gérer des fenêtres "popup".

Lorsque TPFENETR affiche une fenêtre, elle est accessible par toute méthode d'accès rapide à l'écran de TPCRT. TPFENETR définit un système de coordonnées relatives au coin supérieur gauche de la fenêtre. Si votre programme utilise les procédures *Write* et *Writeln*, le texte sera automatiquement déroulé dans la fenêtre.

Lorsque TPFENETR efface une fenêtre, elle restaure non seulement le contenu de l'écran non recouvert par la fenêtre mais aussi la position et la forme du curseur, le système de coordonnées de la fenêtre précédente et les attributs vidéo par défaut. En option, elle sauvegardera également une image de la fenêtre effacée afin que vous puissiez la réafficher sans la réinitialiser.

TPFENETR gère les fenêtres sur une *pile de fenêtres*. Chaque fenêtre est empilée lorsqu'elle est affichée et dépilée lorsqu'elle est effacée. De cette façon, les fenêtres peuvent être effacées seulement en ordre inverse. TPFENETR, néanmoins, fournit une routine spéciale pour déplacer rapidement sur le sommet de la pile une fenêtre intermédiaire.

Dans cette partie, nous utiliserons plusieurs termes qui clarifieront les exposés. Une *fenêtre allouée* est une fenêtre dont l'enregistrement a été alloué (sur le tas) et initialisé avec les attributs et les valeurs de positions de fenêtre par défaut. Une fenêtre allouée peut être visible ou non sur l'écran. Une *fenêtre active* est la fenêtre en cours d'affichage ou la fenêtre temporairement recouverte par une autre. La *fenêtre courante* est la fenêtre la plus récente. Le système de coordonnées des fenêtres ainsi que les attributs vidéo par défaut et la forme du curseur sont définis par la fenêtre courante. Ainsi, les instructions *Write* et *Writeln* écriront toujours dans la fenêtre courante. Une *fenêtre active* peut devenir la *fenêtre courante* de deux manières : en effaçant des fenêtres jusqu'à ce que la fenêtre désirée soit au sommet de la pile ou en appelant la routine (*InitFenSup*) qui permet de placer rapidement la fenêtre désirée sur le sommet de la pile. TPFENETR ne vous permettra pas d'afficher une fenêtre active une seconde fois pour ne pas détruire le buffer de la fenêtre. Cependant, TPFENETR contient des routines qui affichent une nouvelle fenêtre ou placent une fenêtre déjà active sur le sommet de pile.

Dans certains cas, TPFENETR distingue les fenêtres *disjointes* des fenêtres qui se *chevauchent*. Un système de fenêtres disjointes est un système dans lequel aucune fenêtre ne recouvre une autre. Avec les fenêtres disjointes, TPFENETR peut rapidement passer d'une fenêtre à l'autre puisqu'il n'y a pas de zone d'écran à restaurer pendant l'échange. Il suffit simplement de déplacer le curseur, de réinitialiser les coordonnées de fenêtres et de modifier l'attribut texte par défaut. TPFENETR ne sait pas si vous utilisez des fenêtres qui se chevauchent ou des fenêtres disjointes. Il vous appartient donc d'appeler la routine appropriée lorsque vous voulez sélectionner une fenêtre différente. Remarquez cependant, que les fenêtres qui se chevauchent sont toujours plus fiables.

Comme TPECRAN, TPFENETR offre des facilités d'adaptation des couleurs. Vous pouvez alors définir un seul ensemble d'attributs vidéo compatibles avec les moniteurs couleurs et monochromes (*FixeCouleur*, la routine appelée pour adapter les couleurs, est en fait située dans l'unité TPCRT, puisqu'elle est également utilisée par TPMENU).

Les routines qui suivent sont déclarées dans la partie interface de TPFENETR principalement pour l'unité TPMENU qui accéde aux fonctions internes non utilisées dans les applications courantes. La partie description qui suit ne doit pas contenir d'entrée pour :

```
procedure ExplosionCadre(F : PtrDessinFen) ;
   {-explosion d'un cadre de fenêtre}

procedure ImplosionCadre(F : PtrDessinFen) ;
   {-efface une fenêtre qui explose sur l'écran}

procedure SauveCadreOmbre(F : PtrDessinFen) ;
   {-Sauve la partie de l'écran recouverte par la fenêtre ombrée}

procedure DessineCadreOmbre(F : PtrDessinFen) ;
   {-Ombre une fenêtre en modifiant les attributs du texte}

procedure RestaureCadreOmbre(F : PtrDessinFen) ;
   {-restaure la partie de l'écran sous la zone ombrée}
```

TPFENETR : Déclarations

Types

```
EcranV =
   Record
      LignesV : Word ;        {Ligne écran}
      ColsV : Word ;          {Colonne écran}
      SegV : Word ;           {segment où se trouve l'écran}
      FinV : Word ;           {Après le dernier offset valide du segment}
      PtrV : BufP ;           {buffer alloue}
   end ;
```

Les écrans virtuels gérés par TPFENETR sont de type *EcranV*. La partie principale de la mémoire utilisée par un écran virtuel se trouve sur le tas, pointée par le champ *PtrV*. Un écran virtuel a une taille inférieure à un segment de 64 K. N'oubliez pas que chaque position de lignes, de colonnes d'un écran virtuel utilise deux octets, un pour le caractère et un autre pour l'attribut vidéo. Le programme ne doit pas référencer les champs d'un enregistrement *EcranV*. --TPFENETR contient des routines pour manipuler les écrans virtuels.

```
PtrFen = pointer;
```

Chaque variable fenêtre de *Turbo Professionnel* doit être du type *PtrFen*. Une fenêtre occupe 4 octets comme un pointeur dans le segment où la variable fenêtre est déclarée. Après un appel à *InitFen*, la variable *PtrFen* pointe sur une zone de taille variable du tas (environ 40 octets plus 1 ou 2 buffers écran) qui contient toutes les données nécessaires à TPFENETR pour gérer la fenêtre. Bien que TPFENETR exporte seulement ce type de pointeur, vous pouvez vous référer à *EnrFenetre* dans la partie "Implémentation "de l'unité pour voir exactement ce que l'enregistrement contient.

Constantes

```
Explosion : Boolean = false;
```

Positionnez cette valeur à "True" avant d'appeler *InitFen* pour toute fenêtre qui devra exploser avant d'être affichée ou effacée. *InitFen* enregistre la valeur actuelle de *Explosion* dans l'enregistrement de fenêtre et l'utilise pour les affichages. Le cadre et la couleur de fond d'une fenêtre qui explose apparaissent ou disparaissent par étapes et provoquent des effets dynamiques spécifiques.

```
DelaiExplosion : Word = 15 ;
```

Cette valeur définit le délai en millisecondes entre deux explosions. Ajustez-la selon les effets recherchés. Comme les autres constantes globales de TPFENETR *InitFen* enregistre la valeur actuelle de *DelaiExplosion* lorsque la fenêtre est définie.

```
Ombre : Boolean = False ;
```

Positionnez cette valeur à "True" avant d'appeler *InitFen* pour toute fenêtre qui devra être affichée avec une partie ombrée. *InitFen* enregistre la valeur actuelle de *Ombre* dans l'enregistrement fenêtre et l'utilise pour les affichages ultérieurs. Avec *Turbo Professionnel,* les ombres apparaissent toujours sur la droite et au dessous d'une fenêtre. La partie ombrée ajoute deux colonnes et une ligne à la taille de la fenêtre.

```
AttrOmbre : Byte = $07 ;
```

Lorsque TPFENETR affiche une fenêtre ombrée, elle modifie l'attribut vidéo de la partie ombrée avec cet attribut. En sélectionnant un attribut approprié vous pouvez encore lire le texte situé sous la partie ombrée. Si vous voulez masquer totalement le texte recouvert utilisez une valeur de *AttrOmbre* avec les couleurs d'arrière plan et de premier plan identiques. Par exemple, la valeur $77 initialise la partie ombrée avec un blanc pale. Initialisez *AttrOmbre* avant d'appeler *InitFen* qui rangera cette valeur dans l'enregistrement fenêtre pour les affichages ultérieurs.

```
SonAutorise : Boolean = True ;
```

Une fenêtre qui explose peut également produire des sons si *SonAutorise* vaut "True". Remarquez que lorsque les fenêtres explosent, le son est activé par défaut. Nous vous conseillons d'essayer les effets sonores pour vérifier s'ils sont appropriés ou non à une application donnée.

Variables

```
FenActuelle : PtrFen;
```

Cette variable pointe toujours sur la fenêtre actuelle. Si aucune fenêtre n'est affiché, *FenActuelle* vaudra Nil.

ActiveEcranV

Déclaration

```
procedure ActiveEcranV(V : EcranV) ;
```

Objet

Redirige les routines d'écriture rapide sur l'écran virtuel spécifié.

Commentaires

Lorsque vous appelez cette routine, certaines routines d'écriture rapide de TPCRT sont redirigées sur l'écran virtuel spécifié plutôt que sur l'écran physique. Les routines suivantes de TPCRT sont concernées.

EcritVite	TexteRapide	VertVite	RemplitVite
LitVite	LitAttr	ChangeAttr	EcritFacile
CadreFen			

N'utilisez pas les ordres suivants pour écrire sur un écran virtuel :

Write	WriteLn		
EcritViteFen	TexteRapideFen	VertViteFen	RemplitViteFen
CentreVite	CadreDroite	LitViteFen	ChangeAttrFen
LitAttrFen	EcritFacileFen	EcritViteEtCoupe	

Les instructions *Write* et *WriteLn* continueront d'écrire sur l'écran physique. Les autres appels écriront sur l'écran virtuel, mais utiliseront les coordonnées de fenêtre associées à l'écran physique.

Les écrans virtuels de TPFENETR ne supportent pas les coordonnées de fenêtres à l'intérieur de l'écran. Désormais, les coordonnées *Ligne* et *Colonne* spécifiées pour l'une des routine *EcritVite* sont toujours des coordonnées absolues par rapport au coin supérieur gauche de l'écran virtuel.

Les coordonnées de *EcritVite* sont spécifiées en utilisant des variables Byte. Il en résulte que la plus grande coordonnée valide dans l'une des deux dimensions est 255. Pour éviter les erreurs *ActiveEcranV* refusera de rediriger l'écriture sur un écran virtuel si l'une des deux dimensions spécifiées pour l'écran dépasse 255. Dans ce cas, les appels ultérieurs à *EcritVite* continueront à être dirigés sur l'écran physique. La procédure *EcritEcranV* de TPFENETR peut être utilisée pour écrire sur un écran virtuel avec des coordonnées supérieures à 255.

N'écrivez surtout pas des chaînes qui déborderaient du coin inférieur droit d'un écran virtuel. Avec un écran physique, vous n'aurez pas de problème. Avec un écran virtuel la chaîne peut déborder sur le tas et entraîner des erreurs désastreuses. La routine *EcritVite* ne vous protège pas de cette erreur contrairement à la procédure *EcritEcranV*.

Redirigez les routines d'écriture rapide sur l'écran physique en appelant *DesactiveEcranV*. Copiez l'écran virtuel sur l'écran physique en appelant *DeplaceEcrVDansFen*.

Exemple

```
var
   V : EcranV;
begin
   if not CreeEcranV(V, 100, 100) then begin
      WriteLn('Mémoire insuffisante');
      Halt;
   end;
   EffaceEcranV(V, $1E, ' ');
   ActiveEcranV(V);
   EcritVite('Voici l''écran virtuel', 1, 1, $1E);
end.
```

EcritVite n'apparaîtra pas immédiatement sur l'écran physique, mais adressera la ligne 1, colonne 1 de l'écran virtuel V.

Voir aussi

DesactiveEcranV (TPFENETR)
EcritVite (TPCRT)
DeplaceEcrVDansFen (TPFENETR)
EcritEcranV (TPFENETR)

ActiveFen

Déclaration

```
function ActiveFen(F : PtrFen) : Boolean ;
```

Objet

Affiche ou resélectionne une fenêtre chevauchante.

Commentaires

Utilisez cette routine pour afficher une fenêtre chevauchante si cette fenêtre n'est pas déjà sur l'écran. *ActiveFen* appelle *AfficheFen* (si la fenêtre n'est pas active) ou *InitFenSup* (si la fenêtre est active).

Exemples

```
Statut := ActiveFen(A);
Statut := ActiveFen(B);
Statut := ActiveFen(A);
```

Le premier appel à *ActiveFen* ouvre une nouvelle fenêtre A. Le second appel réorganise la pile des fenêtres, et place la fenêtre A sur le sommet qui devient la fenêtre courante.

Voir aussi

AfficheFen (TPFENETR) InitFenSup (TPFENETR)

ActiveFenDisj

Déclaration

```
function ActiveFenDisj(F : PtrFen) : Boolean ;
```

Objet

Affiche ou resélectionne une fenêtre disjointe.

Commentaires

Utilisez cette routine pour afficher une fenêtre disjointe si cette fenêtre n'est pas déjà sur l'écran. *ActiveFenDisj* peut appeler *AfficheFen* (si la fenêtre n'est pas active) ou *InitFenDisjSup* (si la fenêtre est active).

Exemples

```
Statut := ActiveFenDisj(A);
Statut := ActiveFenDisj(B);
Statut := ActiveFenDisj(A);
```

Le premier appel à *ActiveFenDisj* ouvre une nouvelle fenêtre A. Le second appel, avec la fenêtre A en paramètre, repositionne simplement le curseur et réinitialise les coordonnées de la fenêtre.

Voir aussi

AfficheFen (TPFENETR)

InitFenDisjSup (TPFENETR)

AfficheFen

Déclaration

```
function AfficheFen(F : PtrFen) : Boolean
```

Objet

Affiche la fenêtre spécifiée et renvoie la valeur "True" en cas de succès.

Commentaires

Le paramètre *F* spécifie une fenêtre initialisée et allouée précédemment par un appel à *InitFen*. *AfficheFen* enregistre la zone écran recouverte et affiche la fenêtre spécifiée (l'efface éventuellement ou la restaure), initialise la position et la taille du curseur, initialise les attributs vidéo par défaut pour les futures écritures, empile l'enregistrement fenêtre sur la pile active et affecte la nouvelle fenêtre à la variable *FenActuelle*.

Si les variables *Ombre* et *Explosion* valent "True" lorsque la fenêtre *F* a été définie avec *InitFen*, *AfficheFen* sera nécessaire pour créer des effets d'explosion et les ombres.

AfficheFen déplacera le curseur dans le coin supérieur gauche de la fenêtre sauf si l'indicateur *SauveFen* a été initialisé par *InitFen*. Si *SauveFen* a été initialisé et que la fenêtre a été affichée au moins une fois, *AfficheFen* restaurera le contenu de la fenêtre et placera le curseur à la position qu'il avait lorsque la fenêtre a été effacée. *AfficheFen* renverra "False" si la fenêtre spécifiée est déjà active ou s'il n'y a pas assez de place sur le tas pour y ajouter un enregistrement (qui demande 8 octets).

Exemple

```
var Principal : PtrFen;
...
if not InitFen(Principal, 1, 1, 80, 25, True, True, False,
          AttrF, AttrC, AttrE, ' Fenêtre principale ') then {Erreur};
...
if not AfficheFen(Principal) then {Erreur};
```

InitFen initialise la variable fenêtre principale, *AfficheFen* affiche alors *Principal* qui recouvrira tout l'écran. *Principal* comportera un cadre avec l'attribut vidéo *AttrC*, avec l'en-tête 'Fenêtre principale' affiché avec l'attribut *AttrE*. Puisque *EffaceFen* valait "True" et que *SauveFen* valait "False" lors de l'appel à *InitFen*, *AfficheFen* effacera la fenêtre en utilisant l'attribut *AttrF* et placera le curseur dans le coin supérieur gauche de la fenêtre.

Voir aussi

EffaceFenSup (TPFENETR) InitFen (TPFENETR).

ColsEcranV

Déclaration

```
function ColsEcranV(V : EcranV): Word;
```

Objet

Renvoie le nombre de colonnes d'un écran virtuel.

Commentaires

Cette routine renvoie simplement le champ *ColsV* de l'enregistrement de l'écran virtuel. Appelez cette routine au lieu d'utiliser l'enregistrement correspondant à l'écran virtuel car le format interne de cet écran virtuel peut être modifié.

Exemple

```
if not CreeEcranV(V, 25, 80) then begin
   WriteLn('Mémoire insuffisante');
   Halt;
end;
WriteLn('Nombre de colonnes dans l''écran virtuel = ', ColsEcranV(V));
```

Ecrit la valeur 80.

CreeEcranV

Déclaration

```
function CreeEcranV(var V : EcranV ; Ligne, Col : Word) : Boolean ;
```

Objet

Renvoie "True" après avoir alloué un écran virtuel.

Commentaires

Cette fonction initialise un enregistrement d'écran virtuel. Elle vérifie d'abord que le nombre de lignes et de colonnes de l'écran virtuel sont compatibles et que la taille du buffer de l'écran virtuel ne dépasse pas 64 K octets. Puis elle alloue de la place sur le tas pour l'écran virtuel. Si l'une de ces conditions n'est pas satisfaite, *CreeEcranV* renvoie "False" sinon elle renvoie "True".

N'essayez jamais d'écrire sur un écran virtuel avant d'avoir appelé *CreeEcranV*.

Si *Ligne* ou *Col* dépassent 255, la routine *ActiveEcranV* ne pourra pas fonctionner avec l'écran virtuel puisque les routines d'écriture rapide TPCRT interdisent des valeurs supérieures. Utilisez dans ce cas la routine *EcritEcranV*.

L'appel de *CreeEcranV* n'efface pas l'écran virtuel. Cet appel devra donc être suivi d'un appel à *EffaceEcranV*.

Exemples

```
Statut := CreeEcranV(V1, 100, 100);
Statut := CreeEcranV(V2, 256, 100);
Statut := CreeEcranV(V3, 255, 255);
Statut := CreeEcranV(V4, 409, 80);
```

L'écran *V1* occupe environ 20000 octets sur le tas et *EcritVite* l'écrira après l'appel à *ActiveEcranV*. L'écran V2 occupe environ 51200 octets sur le tas et peut être écrit seulement après l'appel à *EcritEcranV*. L'écran V3 ne peut pas être alloué puisqu'il demande 130050 octets sur le tas. L'écran V4 est le plus grand écran possible de 80 colonnes et occupe environ 65520 octets sur le tas.

Voir aussi

ActiveEcranV (TPFENETR) EffaceEcranV (TPFENETR)

DeallouеFen

Déclaration

```
procedure DeallouеFen(F : PtrFen);
```

Objet

Libère la place réservée sur le tas pour la fenêtre spécifiée.

Commentaires

Lorsqu'une fenêtre n'est plus nécessaire, un appel à *DeallouеFen* libèrera l'espace utilisé sur le tas pour cette fenêtre. Cet appel n'est pas nécessaire à la fin du programme puisque le DOS récupère toute la mémoire utilisée par le programme.

Exemple

```
if not InitFen(Principal, 1, 1, 80, 25, True, True, False,
          AttrF, AttrC, AttrE, ' Fenêtre principale ') then {Erreur};
...
DeallouеFen(Principal);
```

La fenêtre principale utilise environ 4070 octets sur le tas (70 octets pour l'enregistrement fenêtre plus 4000 octets pour un seul buffer écran). *DeallouеFen* récupère cet espace.

Voir aussi

InitFen (TPFENETR).

DeplaceEcrVDansFen

Déclaration

```
procedure DeplaceEcrVDansFen(V : EcranV ; Ligne, Col : Word) ;
```

Objet

Recopie le contenu d'un écran virtuel dans la fenêtre en *Ligne, Col.*

Commentaires

Utilisez cette routine pour recopier tout ou partie d'un écran virtuel sur l'écran physique, afin de visualiser l'écran virtuel.

Le coin supérieur gauche de la zone à recopier a pour coordonnées *Ligne, Col* à l'intérieur de l'écran virtuel. *DeplaceEcrVDansFen* prend une partie de l'écran virtuel de la taille de la fenêtre active en cours et la recopie dans la fenêtre actuelle sur l'écran physique. Si la fenêtre actuelle est plus large que l'écran virtuel disponible, commençant en *Ligne, Col*, *DeplaceEcrVDansFen* recopie la zone appropriée de l'écran virtuel dans la fenêtre et ne modifie pas le reste de la fenêtre.

DeplaceEcrVDansFen utilise les variables de TPCRT *WindMin* et *WindMax* pour définir les coordonnées de la fenêtre actuelle. Ces variables sont initialisées par des appels à la procédure *Window* de TPCRT et sont également prises en compte par les nombreuses routines de fenêtrage de TPFENETR. Remarquez que les valeurs de *WindMin* et *WindMax* ne tiennent pas compte des cadres et des ombres des fenêtres gérées par TPFENETR. Par conséquent, *DeplaceEcrVDansFen* remplira seulement la partie centrale de la fenêtre et n'affectera pas le cadre.

En faisant varier les paramètres *Ligne* et *Col* passés à *DeplaceEcrVDansFen* vous pouvez produire des effets de défilement horizontal, vertical ou diagonal sur l'écran physique. Consultez le programme de démonstration DEROULFE sur la disquette de démonstration.

Pour recopier un écran physique sur un écran virtuel, appelez *DeplaceFenDansEcrV*.

Exemples

```
var
   V : EcranV;
   S : String;
   C : Word;
begin
   {Initialise et efface un écran virtuel contenant une ligne
    et 255 colonnes}
   if not CreeEcranV(V, 1, 255) then begin
      Writeln('Mémoire insuffisante');
      Halt;
   end;
   EffaceEcranV(V, $0F, ' ');

   {Ecrit une longue chaîne de caractères dans l'écran virtuel}
   S := '';
   while Length(S) < 255 do
      S := S + 'Ecran virtuels de TPFENETR ';
   EcritEcranV(V, S, 1, 1, $07);

   {Affiche une petite fenêtre dans l'écran physique}
   ClrScr;
   Window(1, 1, 40, 1);

   {Déroule l'écran virtuel sur l'écran physique}
   for C := 1 to 250 do begin
      DeplaceEcrVDansFen(V, 1, C);
      Delay(20);
   end;
end.
```

Simulation de défilement horizontal en utilisant *DeplaceEcrVDansFen* pour recopier un grand écran virtuel dans une fenêtre plus petite.

Voir aussi

DeplaceFenDansEcrV (TPFENETR)

DeplaceFen

Déclaration

```
function  DeplaceFen(XDelta, YDelta : Integer) : Boolean ;
```

Objet

Déplace la fenêtre actuelle d'une distance donnée. Les valeurs positives correspondent à un déplacement vers la droite ou vers le bas.

Commentaires

Cette routine permet de déplacer la fenêtre courante. La fenêtre garde la même taille, mais son contenu est déplacé en même temps.

XDelta et *YDelta* correspondent aux valeurs de déplacement de la fenêtre. Utilisez des nombres négatifs pour déplacer la fenêtre vers la gauche ou vers le haut et des valeurs positives pour la déplacer vers la droite ou vers le bas. Vérifiez que les valeurs de vos déplacements ne font pas sortir la fenêtre de l'écran. En cas de valeurs incorrectes, *DeplaceFen* s'arrêtera et renverra la valeur "False".

DeplaceFen utilise les techniques d'écrans virtuels pour éviter des scintillements d'écran désagréables pendant le déplacement des fenêtres. Pour mener à bien cette tâche, *DeplaceFen* doit allouer temporairement un buffer sur le tas. Le buffer a une taille égale au rectangle qui encadre la fenêtre avant et après le déplacement. Le plus grand buffer possible pour un écran de 25 lignes est égal à 25*80*2=4 000 octets. Si *DeplaceFen* ne peut allouer le buffer, elle renverra un résultat "False".

Une fois que le contenu de la fenêtre a été décalé et repositionné, *DeplaceFen* adapte les coordonnées de la fenêtre courante et la position du curseur. Elle renvoie "True" en cas de succès.

Exemple

```
Statut := InitFen(F, 1, 1, 10, 5, True, True, False, AF, AC, AE, '');
Statut := AfficheFen(F);
Write('Bonjour');
Statut := DeplaceFen(1, 0); Statut := DeplaceFen(0, 1);
Statut := DeplaceFen(-1, -1);
```

Nous définissons ici une petite fenêtre, nous l'affichons, la déplaçons d'une colonne vers la droite, la déplaçons d'une ligne vers le bas puis la restaurons à sa position d'origine. Le programme de démonstration DEROULFE montre les capacités de TPFENETR pour déplacer et recadrer rapidement des fenêtres.

Voir aussi

RecadreFen (TPFENETR)

DeplaceFenDansEcrV

Déclaration

```
procedure DeplaceFenDansEcrV(V : EcranV; Ligne, Col : Word);
```

Objet

Recopie le contenu de la fenêtre actuelle sur l'écran virtuel en *Ligne, Col.*

Commentaires

Cette routine est l'opposé de la procédure *DeplaceEcrVDansFen*. Elle recopie le contenu de la fenêtre actuelle de l'écran physique sur un écran virtuel à partir des coordonnées *Ligne, Col* de l'écran virtuel.

Si la fenêtre actuelle est supérieure à la partie de l'écran disponible commençant en Ligne, Col, *DeplaceFenDansEcrV* recopie la partie appropriée de la fenêtre.

DeplaceFenDansEcrV utilise les variables de TPCRT *WindMin* et *WindMax* pour définir les coordonnées de la fenêtre courante. Ces variables sont également initialisées par des appels à la procédure *Window* de TPCRT et sont aussi traitées par les nombreuses routines de fenêtrage de TPFENETR. Remarquez que les coordonnées dans *WindMin* et *WindMax* ne comportent pas les cadres et ombres des fenêtres gérées par TPFENETR. Par conséquent, *DeplaceFenDansEcrV* recopiera seulement la partie centrale de la fenêtre sans s'occuper du cadre.

Exemple

```
var V : EcranV;
begin
   if not CreeEcranV(V, 25, 80) then begin
      WriteLn('Mémoire insuffisante');
      Halt;
   end;
   {Copie l'écran physique entier sur l'écran virtuel}
   Window(1, 1, 80, 25);
   DeplaceFenDansEcrV(V, 1, 1);
   {Restaure l'écran physique}
   DeplaceEcrVDansFen(V, 1, 1);
end.
```

Alloue un écran virtuel de la même taille que l'écran physique puis recopie tout l'écran physique sur l'écran virtuel. A ce moment, l'écran physique peut être modifié et plus tard restauré par un appel à *DeplaceEcrVDansFen* ou bien l'écran virtuel peut être modifié et affiché rapidement avec l'appel final.

Voir aussi

DeplaceEcrVDansFen (TPFENETR)

DerouleFen

Déclaration

```
procedure DerouleFen(Haut : Boolean; Lignes : Bytes);
```

Objet

Déroule le fenêtre actuelle vers le haut ou vers le bas (*Haut* = "False") du nombre de lignes spécifié.

Commentaires

DerouleFen passe les coordonnées de la fenêtre active avec le paramètre *Lignes* pour *DerouleFenHaut* (si *Haut* = "True") ou *DerouleFenBas* (si *Haut* = "False").

Exemple

```
DerouleFen(True, 1);
```

Avec une fenêtre 80*25, déplacera les lignes 2 à 25 dans les lignes 1 à 24 et effacera la ligne 25.

Voir aussi

DerouleFenBas (TPCRT)

DerouleFenHaut (TPCRT)

DesactiveEcranV

Déclaration

```
procedure DesactiveEcranV ;
```

Objet

Redirige les écritures rapides sur l'écran physique.

Commentaires

Appelez cette routine après avoir appelé *ActiveEcranV* ; les écritures rapides seront redirigées vers l'écran physique.

Voir *ActiveEcranV* pour plus de détails sur la liste des routines d'écritures rapides concernées par cet appel.

Exemple

```
var
   V : EcranV;
begin
   if not CreeEcranV(V, 100, 100) then begin
      Writeln('Mémoire insuffisante');
      Halt;
   end;
   EffaceEcranV(V, $1E, ' ');
   ActiveEcranV(V);
   EcritVite('Voici l''écran virtuel', 1, 1, $1E);
   DesactiveEcranV;
   EcritVite('Voici l''écran physique', 1, 1, $1E);
end.
```

Ecrit d'abord sur l'écran virtuel puis sur l'écran physique.

Voir aussi

ActiveEcranV (TPFENETR)

EcritEcranV

Déclaration

```
procedure EcritEcranV(V : EcranV; C : string;
                      Ligne, Col : Word; Attr : Byte);
```

Objet

Ecrit une chaîne sur l'écran virtuel spécifié.

Commentaires

Cette routine écrit une chaîne directement sur l'écran virtuel. Elle est équivalente à *EcritVite* pour les écrans virtuels. Elle peut être utilisée à la place de *ActiveEcranV* et *EcritVite*. Les paramètres de *EcritEcranV* sont définis comme ceux de *EcritVite*, plus l'écran virtuel (les paramètres *Ligne* et *Col* sont de type Word plutôt que de type Byte).

EcritEcranV a des avantages et des inconvénients par rapport à *ActiveEcranV*. Les avantages : *EcritEcranV* écrit sur l'écran spécifié en un seul appel ; elle peut être dirigée par des appels normaux à *WriteLn / EcritVite* sur l'écran physique ; elle peut écrire sur des écrans virtuels si l'une des dimensions dépasse 255 lignes ou 255 colonnes ; elle contrôle tous les ordres d'écriture pour s'assurer que le buffer de l'écran virtuel n'est pas plein. Les inconvénients : *ActiveEcranV* peut être suivi par des appels aux nombreuses routines autres que *EcritVite*, y compris *TexteRapide, RemplitVite, EcritFacile* etc... Pour plusieurs écritures dans une ligne, *ActiveEcranV* est plus efficace.

Exemple

```
var
   V : EcranV;
begin
   if not CreeEcranV(V, 100, 100) then begin
      WriteLn('Mémoire insuffisante');
      Halt;
   end;
   EffaceEcranV(V, $1E, ' ');
   EcritEcranV(V, 'Voici l''écran virtuel', 1, 1, $1E);
end.
```

Ecrit une chaîne en ligne 1, colonne 1 sur l'écran virtuel.

Voir aussi

ActiveEcranV (TPFENETR) EcritVite (TPCRT)

EcritViteEtCoupe

Déclaration

```
procedure EcritViteEtCoupe(C : string ; Ligne, Col, Attr : Byte);
```

Objet

Ecrit la chaîne C dans la fenêtre actuelle en utilisant les coordonnées de fenêtre relatives et coupe cette chaîne à droite.

Commentaires

EcritViteEtCoupe se superpose à *EcritViteFen* de l'unité TPCRT. Elle utilise les coordonnées relatives et tronque la chaîne affichée pour la cadrer dans les limites de la fenêtre. *EcritViteEtCoupe* n'étend pas l'écriture au delà des limites de la fenêtre comme c'est le cas avec les instructions *Write* et *WriteLn*.

Exemples

```
Statut := InitFen(F, 1, 1, 10, 2, True, True, False, AF, AC, AE, '');
Statut := AfficheFen(F);
EcritViteEtCoupe('une longue phrase dans une petite fenêtre', 1, 1,
                  AF);
```

Puisque la fenêtre courante ne peut contenir que 8 caractères, *EcritViteEtCoupe* tronque la chaîne et affiche seulement 'une long' dans la fenêtre.

Voir aussi

EcritViteFen (TPCRT)

EcritVitePile

Déclaration

```
procedure EcritVitePile(F : PtrFen; C : string;
                        Ligne, Col, Attr : Byte);
```

Objet

Ecrit dans la fenêtre spécifiée, qu'elle soit apparente ou non.

Commentaires

Cette routine fonctionne comme *EcritViteFen* mais comporte une amélioration importante. Elle permet d'écrire sur n'importe quelle fenêtre active même lorsque cette fenêtre est partiellement ou complètement recouverte par des fenêtres qui se trouvent au dessus d'elle dans la pile.

F spécifie la fenêtre à écrire. La fenêtre doit être active sinon *EcritVitePile* s'arrête immédiatement. Si la fenêtre est la fenêtre courante *EcritVitePile* transfère simplement le contrôle à *EcritViteFen*. Sinon *EcritVitePile* utilise les techniques d'écrans virtuels pour dépiler temporairement les fenêtres du dessus, écrire la chaîne, réempiler les fenêtres et transférer le résultat vers l'écran physique.

De ce fait, *EcritVitePile* n'est pas très rapide, surtout lorsqu'il y a beaucoup de fenêtres empilées. De plus, elle produit des effets impressionnants lorsqu'elle est utilisée correctement.

Ligne, Col et *Attr*, sont traités exactement comme avec *EcritViteFen*.

Pour utiliser correctement *EcritVitePile*, toutes les fenêtres actives doivent avoir l'indicateur *SauveFen* positionné à "True" (*SauveFen* est le dernier paramètre booléen spécifié dans un appel à *InitFen*). *EcritVitePile* ne coupe pas les chaînes qui sortent des limites de la fenêtre.

Exemples

```
var Ok : Boolean; W, X : PtrFen; R  : Byte;
begin
   Ok := InitFen(W, 1, 1, 60, 25, True, True, True, $1E, $17, 0, '');
   Ok := InitFen(X, 20, 5, 40, 21, True, True, True, $3F, $30, 0, '');
   Ok := AfficheFen(W); Ok := AfficheFen(X);
   for R := 1 to 23 do
      EcritVitePile(W, ChaineCarac('A', 58), R, 1, $1E);
end.
```

La fenêtre *X* est une petite fenêtre affichée sur une plus grande fenêtre *W*. *EcritVitePile* écrit sur la fenêtre du dessous sans perturber la fenêtre *X*.

Voir aussi

EcritViteEtCoupe (TPFENETR) EcritViteFen (TPCRT)

EffaceEcranV

Déclaration

```
procedure EffaceEcranV(V : EcranV ; Attr : Byte ; Rempl : Char) ;
```

Objet

Efface un écran virtuel avec le caractère et l'attribut spécifiés.

Commentaires

Cette routine remplace la routine *ClrScr* pour des écrans virtuels. Contrairement à la routine habituelle *ClrScr*, vous devez avec cette routine spécifier l'attribut vidéo (*Attr*) et le caractère de remplissage (*Rempl*) comme paramètres de l'écran à effacer. Habituellement le caractère de remplissage est un espace. Cette routine efface toujours l'écran virtuel tout entier.

Comme la routine *CreeEcranV* n'efface pas l'écran qui vient d'être alloué, vous devrez appeler *EffaceEcranV* immédiatement après *CreeEcranV*.

Exemples

```
var V : EcranV;
...
if not CreeEcranV(V, 100, 100) then
   begin
      Writeln('Mémoire insuffisante');
      Halt;
   end;
EffaceEcranV(V, $1E, ' ');
```

Remplit l'écran virtuel avec des espaces, en jaune sur fond bleu.

Voir aussi

CreeEcranV (TPFENETR)

EffaceFenSup

Déclaration

```
Function EffaceFenSup: PtrFen ;
```

Objet

Efface la dernière fenêtre affichée et renvoie son pointeur.

Commentaires

Les fenêtres sont effacées de l'écran dans l'ordre inverse de l'ordre d'affichage.

Lorsqu'une fenêtre est effacée, *EffaceFenSup* exécute plusieurs fonctions. Si l'indicateur *SauveFen* est positionné, le contenu de la fenêtre courante, la forme et la position du curseur sont enregistrés avant que la fenêtre ne soit effacée. Puis le contenu de l'écran recouvert par cette fenêtre est restauré. Enfin, *EffaceFenSup* restaure le curseur, les coordonnées de la fenêtre et les attributs texte de la fenêtre qui était recouverte.

Lorsque *EffaceFenSup* efface la fenêtre courante, elle renvoie un pointeur sur la fenêtre qui vient d'être effacée. Le pointeur est particulièrement pratique lors de l'appel à *DeallouеFen* pour libérer le tas (si la fenêtre n'est plus nécessaire dans le programme). L'appel à *EffaceFenSup* est sans effet lorsque aucune fenêtre n'est affichée, la fonction dans ce cas ne fait rien mais renvoie un pointeur à Nil.

Exemples

```
DeallouеFen(EffaceFenSup);
```

Efface la fenêtre principale et libère le tas.

```
repeat
   F := EffaceFenSup;
until F = Nil;
```

Efface toutes les fenêtres actives qui restent allouées pour une utilisation future.

Voir aussi

DeallouеFen (TPFENETR).

FenEstActive

Déclaration

```
function FenEstActive(F : PtrFen) : Boolean;
```

Objet

Renvoie "True" si la fenêtre spécifiée est la fenêtre active courante.

Commentaires

Cette fonction est très utile pour les systèmes de menus avec fenêtres. Si un utilisateur demande l'affichage d'une fenêtre particulière, le programme peut décider s'il doit utiliser *InitFenSup* ou *AfficheFen* selon que la fenêtre demandée est déjà active ou non.

Exemple

```
if FenEstActive(F) then
   Statut := InitFenSup(F)
else
   Statut := AfficheFen(F);
```

Cette séquence évite d'afficher deux fois la même fenêtre.

Voir aussi

AfficheFen (TPFENETR) InitFenSup (TPFENETR).

InitCadreInactif

Déclaration

```
procedure InitCadreInactif(F : PrtFen; TC : TabCadre;
                           AttrCadre, EnteteAttr : Byte);
```

Objet

Définit l'attribut et le type du cadre pour les fenêtres visibles mais pas courantes.

Commentaires

TPFENETR permet de définir automatiquement les différents cadres d'une fenêtre. Par défaut, les deux cadres sont équivalents. Appelez la routine *InitCadreInactif* pour définir les différents caractères et les couleurs de cadre pour les fenêtres actives qui ne sont pas en cours. Ensuite, TPFENETR utilise automatiquement les attributs inactifs lorsqu'une autre fenêtre est affichée ou placée sur sommet de pile.

Exemple

```
CaracsCadre := #201#200#187#188#205#186;
Statut:=InitFen(A, 01, 1, 40, 25, True, True, False, AF, $0F, $0F, '');
Statut:=InitFen(B, 41, 1, 80, 25, True, True, False, AF, $0F, $0F, '');
InitCadreInactif(A, #218#192#191#217#196#179, $07, $07);
InitCadreInactif(B, #218#192#191#217#196#179, $07, $07);
Statut := AfficheFen(A);
Statut := AfficheFen(B);
```

La fenêtre courante est affichée avec un double cadre et $0F comme couleur de cadre. Les fenêtres actives qui ne sont pas en cours sont affichées avec un cadre simple et un attribut de cadre $07. Lorsque cette séquence est exécutée, la fenêtre A aura un cadre simple et la fenêtre B un cadre double.

Voir aussi

AfficheFen (TPFENETR)

InitFen

Déclaration

```
function InitFen(Var F : PtrFen; XBas, YBas : Byte;
                 XHaut, YHaut : Byte; DessinCadre : Boolean;
                 EffFen : Boolean; SauveFen : Boolean;
                 AttrFen : Byte;  AttrCadre : Byte;
                 AttrEntete : Byte;   Entete : String) : Boolean ;
```

Objet

Alloue et initialise une nouvelle fenêtre sans l'afficher.

Commentaires

InitFen définit les caractéristiques de chaque nouvelle fenêtre. Appelez cette fonction pour allouer et initialiser une variable fenêtre avant de l'afficher.

Le paramètre *F* est la fenêtre à initialiser. Au retour, il contiendra un pointeur sur la structure des données relatives à la fenêtre.

XBas, YBas, XHaut et *YHaut* définissent les coins supérieur gauche et inférieur droit de la fenêtre. Les coordonnées *X* appartiennent à l'intervalle *1..LargeurEcran* et les coordonnées *Y* à l'intervalle *1..HauteurEcran* plus 1. Puisque *InitFen* ne contrôle pas la validité des coordonnées, si la fenêtre comporte un cadre, les coordonnées de fenêtre doivent inclure les positions utilisées par le cadre. Quant une fenêtre est affichée, le système de coordonnées relatives définira la position 1,1 juste à l'intérieur du coin supérieur gauche du cadre.

Initialisez le paramètre *DessinCadre* à "True" si vous voulez que TPFENETR dessine un cadre autour de la fenêtre lorsqu'elle est affichée. Les caractères utilisés pour le dessin de cadre sont les caractères actuels de la variable *CaracsCadre* de TPCRT lorsque *InitFen* est appelée. Vous pouvez spécifier un autre ensemble de caractères pour le cadre en les affectant directement à *CaracsCadre* ou en appelant la procédure *InitCaracsCadre* de TPCRT. Le cadre sera dessiné avec l'attribut vidéo spécifié par *AttrEntete*.

Initialisez le paramètre *EffFen* à "True" si vous voulez que TPFENETR efface le corps de la fenêtre lorsqu'elle est affichée. La fenêtre sera remplie par des blancs avec l'attribut vidéo *AttrFen*.

En initialisant le paramètre *SauveFen* à "True", TPFENETR sauvegardera le contenu actuel de cette fenêtre avant de l'effacer. Lorsque la fenêtre sera affichée, TPFENETR la restaurera. Le paramètre *SauveFen*, s'il vaut "True", a priorité sur l'indicateur *EffFen* sauf la première fois que la fenêtre est affichée.

Si vous voulez centrer une chaîne de caractères dans la barre supérieure du cadre de la fenêtre, spécifiez une chaîne pour le paramètre *Entete*. Sinon, passez une chaîne vide à cette position. La chaîne d'en-tête sera affichée avec l'attribut vidéo *AttrEntete*.

La forme initiale du curseur pour une fenêtre sera celle en vigueur lorsque *InitFen* est appelée. Ainsi, si le curseur est caché lorsqu'une nouvelle fenêtre est appelée, le curseur par défaut pour cette fenêtre sera également caché. Lorsque la fenêtre est active, le curseur peut être modifié en utilisant n'importe quelle routine de TPCRT.

InitFen alloue de la place sur le tas pour l'enregistrement de fenêtres et pour les buffers de l'écran. L'enregistrement fenêtre lui-même occupe environ 40 octets. La taille du buffer écran est fonction des coordonnées de la fenêtre :

```
Taille buffer = 2 * (XHaut - XBas + 1) * (YHaut - YBas + 1) octets.
```

Si le paramètre *SauveFen* vaut "True", deux buffers de cette taille seront alloués sinon un seul buffer sera alloué. *InitFen* alloue un espace supplémentaire pour traiter la chaîne en-tête si elle existe. *InitFen* retournera "False" s'il n'est pas possible d'allouer des blocs sur le tas.

InitFen prend également en compte plusieurs constantes globales typées qui sont enregistrées dans l'enregistrement de fenêtre. Les constantes *Ombre, AttrOmbre, Explosion, DelaiExplosion* et *SonAutorise*, concernent tous les affichages éventuels de fenêtres. Vérifiez que ces constantes ont une valeur correcte avant d'appeler *InitFen*.

Remarquez que *InitFen* n'affiche pas elle-même une fenêtre. Pour afficher la fenêtre, utilisez *AfficheFen*. Pour libérer une fenêtre, utilisez *DeallouеFen*.

Exemples

```
var Principal, Statut : PtrFen;
...
if not InitFen(Principal, 1, 1, 80, 25, True, True, False,
          AttrF, AttrC, AttrE, ' Fenêtre principale ') then {Erreur};
```

Alloue une fenêtre plein-écran encadrée initialement vide et qui ne sera pas sauvegardée. Le titre de la fenêtre sera "Fenêtre principale"

```
if not InitFen(Statut, 10, 10, 30, 15, True, True, True,
               AttrF, AttrC, AttrE, 'Statut') then {Erreur};
```

Alloue une petite fenêtre. Puisque l'indicateur *SauveFen* vaut "True", le contenu de cette fenêtre sera sauvegardé et restauré lorsque la fenêtre sera effacée et réaffichée.

Voir aussi

Déclarations (TPCRT)

DeallouеFen (TPFENETR)

InitFenDisjSup

Déclaration

```
function InitFenDisjSup(F : PtrFen): Boolean;
```

Objet

La fenêtre disjointe déjà activée devient la fenêtre courante.

Commentaires

Cette routine est applicable seulement lorsque deux ou plusieurs fenêtres disjointes (non chevauchantes) sont affichées comme fenêtres courantes. Dans ce cas, un appel à *InitFenDisjSup* active la fenêtre spécifiée comme fenêtre courante et adapte les coordonnées de fenêtre, la position du curseur et les valeurs de couleurs de texte par défaut.

InitFenDisjSup renverra "False" si la fenêtre spécifiée n'est pas la fenêtre actuelle affichée. Comme *InitFenDisjSup* a besoin d'un petit espace sur le tas lorsqu'elle réorganise la pile des fenêtres, elle renverra également "False" s'il n'y a pas assez de place sur le tas.

Cette routine est plus rapide que *InitFenSup* mais doit utiliser un écran virtuel pour effacer et redessiner les éventuelles fenêtres qui se chevauchent.

Exemple

```
Statut := InitFen(A, 01, 1, 40, 25, True, True, False, AF, AC, AE, '');
Statut := InitFen(B, 41, 1, 80, 25, True, True, False, AF, AC, AE, '');
Statut := AfficheFen(A);
Statut := AfficheFen(B);
Statut := InitFenDisjSup(A)
```

Deux fenêtres qui recouvrent tout l'écran sans se chevaucher sont définies et affichées et, *InitFenDisjSup* rend le contrôle à la première fenêtre.

Voir aussi

ActiveFenDisj (TPFENETR)

InitFenSup (TPFENETR)

InitFenSup

Déclaration

```
Function InitFenSup(F : PtrFen) : Boolean;
```

Objet

Place la dernière fenêtre active sur le sommet de la pile et renvoie "True" si l'opération se termine bien.

Commentaires

Un inconvénient inhérent à la gestion des fenêtres par pile est que les fenêtres qui ne sont pas des fenêtres actuelles ne sont pas accessibles immédiatement. La fonction *InitFenSup* pallie à cet inconvénient en fournissant une méthode permettant d'activer n'importe quelle fenêtre comme fenêtre courante.

Spécifiez dans le paramètre *F* la fenêtre que vous voulez traiter comme fenêtre actuelle.

Cette fenêtre doit être déjà active sinon *InitFenSup* s'arrêtera et renverra la valeur "False".

Si la fenêtre est active, *InitFenSup* effacera les fenêtres du dessous, y compris la fenêtre à traiter et les réaffichera dans l'ordre avec la fenêtre requise sur le dessus.

Si la fenêtre requise est déjà la fenêtre courante, *InitFenSup* s'arrête mais renvoie la valeur "True".

Cette méthode n'est possible que si l'indicateur *SauveFen* vaut "True" pour toutes les fenêtres actives. Sinon les contenus des fenêtres réempilées seront perdus.

La routine *InitFenSup* ne vérifie pas que l'indicateur *SauveFen* vaut "True" pour toutes les fenêtres.

Lancez le programme de démonstration DEROULFE pour voir fonctionner la routine *InitFenSup*.

Notez que *InitFenSup* utilise les techniques d'écrans virtuels pour masquer toutes les étapes d'effacement et de restructuration des fenêtres. Pour mener à bien cette tâche, *InitFenSup* doit allouer un buffer temporaire sur le tas de taille égale à l'écran physique, c'est-à-dire 25*80*2=4 000 octets. Si cette quantité de mémoire (plus un espace supplémentaire pour la manipulation de pile) n'est pas disponible, *InitFenSup* renverra "False".

Exemple

```
if not AfficheFen(A) then {erreur};
if not AfficheFen(B) then {erreur};
if not InitFenSup(A) then {erreur};
```

L'instruction *InitFenSup* empile la fenêtre *A* sur le sommet de la pile.

Voir aussi

InitFen (TPFENETR)
InitFenDisjSup (TPFENETR)
FenEstActive (TPFENETR)

LibereEcranV

Déclaration

```
procedure LibereEcranV(Var V : EcranV) ;
```

Objet

Libère l'espace occupé par un écran virtuel.

Commentaires

Cette routine libère la partie du tas alloué pour l'écran virtuel *V* par *CreeEcranV*. Notez que vous ne pouvez pas utiliser l'écran virtuel *V* après cet appel, puisque *LibereEcranV* remplit tous les champs de l'enregistrement V avec des zéros.

Exemples

```
var
   V : EcranV;
begin
   if not CreeEcranV(V, 100, 100) then begin
      Writeln('Mémoire insuffisante');
   Halt;
   end;
   EffaceEcranV(V, $1E, ' ');
   ActiveEcranV(V);
   EcritVite('Voici l''écran virtuel', 1, 1, $1E);
   DesactiveEcranV;
   EcritVite('Voici l''écran physique', 1, 1, $1E);
   LibereEcranV(V);
end.
```

Voir aussi

CreeEcranV (TPFENETR)

LignesEcranV

Déclaration

```
function LignesEcranV(V : EcranV): Word;
```

Objet

Renvoie le nombre de lignes d'un écran virtuel.

Commentaires

Cette routine renvoie simplement le champ *LignesV* de l'enregistrement de l'écran virtuel. Appelez cette routine au lieu d'utiliser l'enregistrement correspondant à l'écran virtuel au cas où le format interne de cette écran virtuel viendrait à changer.

Exemple

```
if not CreeEcranV(V, 25, 80) then begin
   WriteLn('Mémoire insuffisante');
   Halt;
end;
WriteLn('Nombre de Lignes dans l''écran virtuel = ', LignesEcranV(V));
```

Ecrit la valeur 80.

LitEcranV

Déclaration

```
procedure LitEcranV(V : EcranV; Nombre : Byte;
                    Ligne, Col : Word; var C : string);
```

Objet

Lit *Nombre* caractères sur l'écran virtuel à partir de *Ligne, Col.*

Commentaires

Cette routine lit une chaîne directement sur l'écran virtuel. Elle est équivalente à *LitVite* pour les écrans virtuels. Elle peut être utilisée à la place de *ActiveEcranV* et *LitVite*. Les paramètres de *LitEcranV* sont définis comme ceux de *LitVite* plus la spécification de l'écran virtuel à lire (mais les paramètres *Ligne* et *Col* sont de type Word plutôt que de type Byte).

Exemple

```
var
   V : EcranV;
   S : String;
begin
   if not CreeEcranV(V, 100, 100) then begin
   WriteLn('Mémoire insuffisante');
   Halt;
   end;
   EffaceEcranV(V, $1E, ' ');
   EcritEcranV(V, 'Voici l''écran virtuel', 1, 1, $1E);
   LitEcranV(V, 20, 1, 1, S);
end.
```

Voir aussi

ActiveEcranV (TPFENETR)
EcritEcranV (TPFENETR)
LitVite (TPCRT)

RecadreFen

Déclaration

```
function RecadreFen(XDelta, YDelta : Integer; Rempl : Char): Boolean;
```

Objet

Recadre la fenêtre courante. Les valeurs positives indiquent le déplacement vers la droite ou vers le bas.

Commentaires

Le coin supérieur gauche de la fenêtre restera à la même position mais les bords inférieurs droits seront déplacés pour prendre en compte la nouvelle taille.

XDelta et *YDelta* correspondent aux valeurs d'ajustement de la fenêtre. Utilisez des nombres négatifs pour rapetisser la fenêtre et des nombres positifs pour l'agrandir. Vérifiez bien que la nouvelle fenêtre tient entièrement sur l'écran sinon *RecadreFen* s'arrêtera et renverra la valeur "False".

Lorsque le fenêtre se déplace, *RecadreFen* remplit les nouvelles zones de la fenêtre avec le caractère *Rempl* que vous avez spécifié. En général, ce caractère est l'espace.

RecadreFen utilise les techniques de l'écran virtuel pour éviter des scintillements indésirables de l'écran pendant la modification de la fenêtre. Pour cela, elle doit allouer temporairement un buffer sur la tas qui devra contenir la fenêtre avant et après le recadrage. La plus grande valeur possible de buffer pour un écran de 25 lignes et 25*80*2=4 000 octets. Si *RecadreFen* ne peut allouer le buffer, elle renvoie la valeur "False".

Une fois le recadrage effectué, *RecadreFen* adapte les coordonnées de la fenêtre courante et déplace le curseur dans la fenêtre si nécessaire. Elle renvoie la valeur "True" en cas de succès.

Un recadrage effectué par petites étapes tend à fragmenter le tas car TPFENETR libère et alloue les buffers de fenêtre pour adapter la nouvelle taille. Pour minimiser cette fragmentation, TPFENETR modifie la taille des buffers de fenêtre par multiples de 2, ce qui diminue le nombre de réallocations.

Exemple

```
Statut := InitFen(F, 1, 1, 10, 5, True, True, False, AF, AC, AE, '');
Statut := AfficheFen(F);
Write('Bonjour');
Statut := RecadreFen(0, 15, ' ');
Statut := RecadreFen(0, 15, ' ');
```

Le premier appel à *RecadreFen* augmente la hauteur de la fenêtre de 5 lignes (1..5) à 20 lignes (1..20). Le second appel à *RecadreFen* renvoie la valeur de statut "False" puisque les 35 lignes de la fenêtre ne se cadrent pas normalement sur l'écran.

Le programme de démonstration DEROULFE montre les capacités de TPFENETR pour déplacer et recadrer des fenêtres rapidement.

Voir aussi

DeplaceFen (TPFENETR)

TPMENU : Menus déroulants

Les menus déroulants à fenêtres multiples sont aujourd'hui banalisés sur les applications fonctionnant sur micro-ordinateurs. Comme beaucoup d'entre nous le savent, la réalisation d'une interface peut prendre plus de temps que l'écriture du code de l'application elle-même.

TPMENU est un ensemble de routines puissantes permettant de gagner du temps sur le développement de systèmes de menus sophistiqués.

Les menus peuvent être de styles variés. Les menus déroulants de *Borland* appartiennent à un style de menus couramment utilisé. Les menus classiques *Lotus* composés d'une barre lumineuse balayant une ligne horizontale appartiennent à un autre style.

TPMENU peut émuler facilement ces deux styles et permet de définir d'autres styles en fonction de votre fantaisie. TPMENU intègre pleinement les dernières innovations : fenêtres ombrées, fenêtres qui explosent, menus avec "hot key" et sélection avec souris.

Les structures de données sont totalement dynamiques. La capacité d'un système de menus donné est seulement limitée par la place disponible sur le tas. TPMENU est composé d'un ensemble complet de listes et d'arbres dynamiques. Heureusement, vous n'avez pas à connaître tous les détails. TPMENU exporte des routines de haut niveau, ce qui la rend plus facile à utiliser.

La meilleure façon est peut être d'introduire TPMENU avec un exemple simple :

```
program  DemoMenu;
uses
   TpCrt,
   TpMenu;
var
   Ch : Char;
   Principal : Menu;
   Touche: ToucheMenu;

procedure InitialiseMenu(var M : Menu);
const
   Couleur1 : TabCouleurMenu  = ($0E,$2E,$03,$1E,$0B,$0E, $19,$78);
   Cadre1   : TabCadre = #201#200#187#188#205#186;
   Cadre2   : TabCadre = #203#200#203#188#205#186;
```

```
begin
   M := NouvMenu([],nil);
   SousMenu(1,2,1,Horizontal,Cadre1,Couleur1,'TPMENU');
      ElemMenu('Premier',2,1,1,'');
      SousMenu(2,4,1,Vertical,Cadre2,Couleur1,'');
         ElemMenu('Alpha',1,1,2,'');
         ElemMenu('Beta',2,1,3,'');
         LanceSousNiveau;
      ElemMenu('Second',12,1,4,'');
      SousMenu(11,4,1,Vertical,Cadre2,Couleur1,'');
         ElemMenu('Gamma',1,1,5,'');
         ElemMenu('Delta',2,1,6,'');
         LanceSousNiveau;
      LanceSousNiveau;
   InitMenu(M);
end;

begin
   InitialiseMenu(Principal);
   repeat
      Touche:= ChoixMenu(Principal,Ch);
      GotoXY(1,25);
      write('Sélection = ',Touche,' avec la touche de code ',ord(Ch));
      Delay(1000);
      EffaceMenu(Principal,False);
   until (Ch = ^M) and (Touche = 6);
end.
```

La procédure *InitialiseMenu* initialise le système de menus qui est dans ce cas un système de menus déroulants à deux niveaux. Le code source de *InitialiseMenu* a été généré par le programme utilitaire CREEMENU.

Vous noterez que *InitialiseMenu* appelle cinq routines de TPMENU. *NouvMenu* initialise un nouveau système de menus vides et définit certaines propriétés de ses variables. *SousMenu* ajoute un nouveau sous-menu (une zone rectangulaire de l'écran contenant un ou plusieurs éléments de menus) au noeud courant du système de menus. Chaque sous-menu est défini par une position, une orientation, un type de cadres, un ensemble de couleurs, un titre et une ligne d'aide. *ElemMenu* ajoute une nouvelle ligne d'options sur le sous-menu actuel.

Une option de menu est définie par son nom, sa position à l'intérieur du sous-menu, la touche utilisée pour la sélectionner et un code renvoyé quand l'option est sélectionnée et une chaîne d'aide facultative. *LanceSousNiveau* indique à TPMENU que le sous-menu actuel est terminé et que le niveau précédent doit être activé. Avec *InitMenu* l'option sélectionnée initialement est la première de chaque sous-menu.

La boucle principale du programme exemple appelle *ChoixMenu* qui est la routine principale de TPMENU. *ChoixMenu* affiche le système de menus et permet à l'utilisateur de le balayer en utilisant les touches du curseur, (ENTER) et (ESC) et de renvoyer l'option sélectionnée. L'option sélectionnée par l'utilisateur est renvoyée en deux parties. La touche de fonction utilisée pour la sélectionner est un code unique pour l'option sélectionnée. La touche de fonction peut être nécessaire pour différencier une demande d'aide et une demande d'exécution de fonction ou même pour supporter des "HotKey" permettant d'accéder à distance à une partie d'un menu. Le code unique est celui affecté pendant l'initialisation du menu.

Votre programme peut choisir la fonction à exécuter en fonction de ce code.

La variable *NumCmdMenu* vous aidera à faire des choix. C'est une variable globale de type *TMtype* qui contient la dernière commande exécutée par l'utilisateur (voir les déclarations ci-dessous). Dans certains cas, cette donnée offre un moyen plus sûr pour déterminer l'action à exécuter, notamment dans le cas où le support de la souris de TPMENU est activé mais aussi lorsque l'utilisateur fournit les moyens de modifier le rôle des touches (voir appendice C). En général, nous vous recommandons donc d'utiliser *NumCmdMenu* plutôt que le code de la touche renvoyée par *ChoixMenu*.

Au retour de *ChoixMenu*, le système de menus restera sur l'écran. Le programme peut appeler la routine *EffaceMenu* de TPMENU pour restaurer l'écran.

TPMENU exporte également des routines pour lire, écrire et compacter des bibliothèques de menus sur disquettes.

Un exemple d'utilisation de TPMENU est donné dans le programme de démonstration CREEMENU. CREEMENU est un outil puissant qui vous aidera à définir des systèmes de menus hiérarchiques et qui génèrera le code source.

Les routines suivantes sont déclarées dans la partie interface de TPMENU pour l'utilisation exclusive du programme utilitaire CREEMENU, nécessitant des accès aux fonctions internes qui ne sont pas nécessaires dans des applications normales.

La description qui suit ne contient pas d'entrées pour :

```
Procedure EffaceFen (XB, YB, XH, YH, Attr : Byte);
   {efface une partie de l'écran avec l'attribut spécifié}

Procedure DeallouéSousMenu(Var SousMenu : SousMenuP);
   {Libère un sous-menu et ses descendants}

Procedure DessineCadre (XB, YB, XH, YH Attr : Byte ;Cadre  : TabCadre);
   {dessine un cadre autour d'une fenêtre}

Procedure DessineElement(SousMenu:SousMenuP; Element:ElemP;
                         UtlFonc:Pointer);
   {dessine une option dans un sous-menu}

Procedure DessineSousMenu  (SousMenu : SousMenuP ; UtlFonc : Pointer);
   {dessine un sous-menu sur l'écran}

Procedure EffaceSousMenu (SousMenu : SousMenuP);
   {efface un sous-menu de l'écran}

Procedure DepileSousMenu (Mnu : Menu );
   {supprime un sous-menu de la pile active}

Procedure EmpileSousMenu (Mnu : Menu ; SousMenu : SousMenuP);
   {place un sous-menu sur la pile active du menu}
```

TPMENU : déclarations

Constantes

```
DeltaCadre : array[Boolean] of byte = (1,0) ;
```

Cette constante permet à TPMENU de corriger les coordonnées pour les menus avec et sans cadre. Les applications ne devraient pas avoir besoin de l'utiliser.

```
CacheCurseur : Boolean = True;
```

Positionnez cette constante à "False" avant d'appeler *ChoixMenu*, si vous désirez que le clignotement du curseur reste visible pendant l'affichage du menu.

```
CadreLotus= #255#255#255#255#255#255;
SansCadre = CadreLotus;
```

Les menus de styles Lotus sont des cas particuliers pour TPMENU car ils ne possèdent pas de cadre autour de chaque menu. Pour satisfaire cette condition, *TabCadre* est définie pour Lotus. Les applications utilisant des menus sans cadre devront spécifier la constante *CadreLotus* toutes les fois que *TabCadre* est appelée.

```
TMRien = 0;          {N'est pas une commande}
TMAlpha = 1;         {Caractère alphanumérique}
TMHaut = 2;          {Curseur vers le haut}
TMBas = 3;           {Curseur vers le bas}
TMGauche = 6;        {Curseur à gauche}
TMDroite = 7;        {Curseur à droite}
TMQuitte = 8;        {Sortie du menu}
TMSelecte = 9;       {Sélectionne l'élément courant}
TMAide = 10;         {Fournit l'aide sur l'élément courant}
TMHome = 11;         {Curseur sur le premier élément du menu}
TMEnd = 12;          {Curseur sur le dernier élément du menu}
TMSonde = 13;        {Sélection d'élément avec la souris}
TMUtl0 = 14;         {Commande de sortie utilisateur}
....
TMUtl3 = 17;
```

Ce sont les codes pour chacune des commandes de sélection et les déplacements du curseur de TPMENU. La plupart d'entre elles s'expliquent d'elles-mêmes. La commande *TMAide* est activée lorsque vous pressez (F1) ou lorsque les deux boutons de la souris sont relâchés. Si une routine d'aide utilisateur a été spécifiée, elle sera appelée. La commande *TMSonde* a été ajoutée spécialement pour être compatible avec la souris. Cette commande est activée toutes les fois que le bouton gauche de la souris est relâché. TPMENU vérifie alors que le curseur de la souris est aligné avec un élément de menu valide et si oui, le sélectionne.

Les quatres derniers (*TMUtl0..TMUtl3*) permettent de définir des points de sortie utilisateur. Si l'une de ces commandes est exécutée, TPMENU se comportera comme si une "hot key" était pressée. Elle peut alors contrôler la variable globale *NumCmdMenu* et exécuter la fonction associée à la commande. Pour plus de détails sur l'implémentation des commandes utilisateur, référez-vous à l'appendice C.

```
SansAide = 0;
```

Chaque menu d'un système TPMENU doit comporter une ligne de texte d'aide. Si vous ne voulez pas de ligne d'aide, spécifiez *SansAide* pour la ligne d'aide, afin de diminuer le temps de recherche et l'allocation mémoire.

Types

```
TypeEtatMenu    =                    {état d'une opération de menus}
(SuccesMenu,                         {opération bien terminée}
MemMenuInsuf,                        {mémoire insuffisante}
FichMenuIntrouvable,                 {le fichier de la bibliothèque
                                      menus est introuvable}
FichMenuNonBiblio                    {le fichier n'est pas un fichier
                                      bibliothèque}
MembreBiblioMenuIntrouvable          {un élément de bibliothèque
                                      spécifié n'est pas trouvé}
ErrLecFichMenu,                      {erreur pendant la lecture d'un
                                      fichier  menu}
ErrEcrFichMenu,                      {erreur pendant l'écriture d'un
                                      fichier menu}
CreationFichMenuImpossible,          {création du fichier bibliothèque
                                      impossible}
FichMenuEndommage,                   {le fichier menu est endommagé}
BiblioMenuPleine),                   {il n'y a plus de place dans
                                      l'index de labibliothèque pour
                                      ajouter une nouvelle entrée}
```

TypeEtatMenu définit les différentes valeurs renvoyées par la fonction *EtatMenu*. Toutes autres valeurs que *SuccesMenu* indique qu'une erreur s'est produite.

```
EnsCaracMenu = set of Char;
```

TPMENU exporte ce type pour permettre la spécification d'un ensemble de caractères de sortie des menus. Toutes les fois que l'utilisateur appuie sur l'un des caractères appartenant à cet ensemble, la fonction *ChoixMenu* sortira du niveau actuel et retournera au niveau actuel précédent.

```
ToucheMenu = LongInt;
```

Chaque option du menu est associée à une touche. La touche a une valeur unique qui identifie le choix. TPMENU définit la touche comme un type de données de 4 octets. En utilisant le changement de type, une application

peut enregistrer et charger des types énumérés, des entiers, des entiers longs ou des pointeurs à l'aide des touches. Si l'adresse d'une donnée ne tient pas sur 4 octets, un pointeur sur une structure de données plus grande devrait être enregistré dans *ToucheMenu*. En général, nous vous recommandons d'attribuer des valeurs à *ToucheMenu* pour chaque élément qui soient comprises entre 1 et 10 000. De cette façon, les valeurs de *ToucheMenu* sont compatibles avec l'index d'aide utilisé par l'unité TPAIDE. L'utilitaire CREEMENU générera automatiquement des touches à partir du numéro 1 et les incrémentera pour chaque élément.

```
Orientation= (Vertical, Horizontal, SansOrient) ;
```

Chaque sous-menu dans un système de menus doit avoir l'une des deux orientations : verticale ou horizontale. La valeur *SansOrient* est utilisée en interne pour dérouter les erreurs et ne devrait pas être utilisée dans une application.

```
TypeCouleurMenu =   {Couleurs utilisées par le système de menus}
   (CouleurCadre,   {Couleurs utilisées pour les cadres de menus}
    CouleurEntete,  {Couleurs utilisées pour la chaîne entête dans
                     un cadre}
    CouleurCorps,   {Couleurs utilisées pour le corps d'un menu}
    CouleurSelect,  {Couleurs utilisées pour l'élément sélectionné}
    CouleurSurbrillance,    {Couleurs utilisées pour le caractère pick
                             de chaque  élément}
    CouleurAide             {Couleurs utilisées pour la ligne d'aide
                             éventuellement}
   {$IFDEF Tpro5Menu}
      ,
      CouleurDesactivee,    {Couleurs utilisées pour désactiver
                            temporairement des éléments}
      CouleurOmbre          {Couleurs utilisées pour les ombres de
                             menus}
   {$ENDIF}
   );
TabCouleurMenu = array[TypeCouleurMenu] of byte;
```

TypeCouleurMenu définit les différentes couleurs utilisées dans un système de menus. *CouleurCadre* est utilisée pour le cadre entourant un menu. *CouleurEntete* est utilisée pour le titre du menu centré dans la barre supérieure du cadre. *CouleurCorps* définit la couleur du fond du menu et de toutes les options non sélectionnées. *CouleurSelect* définit la couleur de l'option actuellement sélectionnée dans un menu. *CouleurSurbrillance* s'applique à l'initiale utilisée pour sélectionner chaque option. Finalement, *CouleurAide* est l'attribut utilisé pour la ligne d'aide facultative.

Avec *Turbo Professionnel* version 5, deux attributs supplémentaires sont définis. *CouleurDesactivee* définit la couleur des élément de menus qui sont temporairement désactivés et ainsi ne peuvent plus être sélectionnés. *CouleurOmbre* est l'attribut à utiliser pour les encadrés ombrés à droite et en dessous d'un sous-menu. L'ombre est sélectionnée par un mécanisme séparé, aussi *CouleurOmbre* sera ignorée dans certaines situations. *TabCouleurMenu* permet de rassembler ces couleurs en une seule structure.

```
Menu = ^EnrMenu;
```

Une variable de ce type est l'interface principale entre votre programme et le système de menus. TPMENU initialise une variable de type *Menu* lorsque vous appelez la fonction *NouvMenu* et poursuit la construction étagée de la structure par des appels successifs à *SousMenu* et *ElemMenu*. Vous pouvez alors passer une variable de type *Menu* à la routine *ChoixMenu* qui affiche le système de menus et permet à l'utilisateur de sélectionner un élément. Voir TPMENU.IN0 pour plus de détails sur le type *EnrMenu*.

```
TMType = TMRien..TMUtl3;
```

TMType définit la classe des actions exécutées par le système de menus. Elle comporte des actions telles que curseur en haut et curseur en bas, sélection d'élément et sortie de menu. Voir les constantes ci-dessus pour plus de détails.

Variables

```
PtrToucheMenu : Pointer;
```

Pointe sur une routine qui renverra la prochaine touche. Voir l'appendice pour plus de détails.

```
PrtAideMenu : Pointer;
```

Pointe sur une routine appelée lorsque la commande d'aide est exécutée. Voir l'appendice pour plus de détails.

```
NumCmdMenu : TMType;
```

Dernière commande de menu donnée à *ChoixMenu*. Elle peut être utilisée pour déterminer si la sortie utilisateur se fait par une sélection normale (ENTER), une sélection alphabétique ou une "hotkey" (*NumCmdMenu* = *TMSelecte*), avec une commande de sortie utilisateur (*NumCmdMenu* = *TMUtl0..TMUtl3*) ou par la frappe de ESC au niveau primaire de menu (*NumCmdMenu* = *TMQuitte*). Puisque les touche associées à chaque commande de menu sont possibles, il est souvent préférable d'évaluer *NumCmdMenu* après avoir appelé *ChoixMenu* plutôt qu'en consultant le paramètre *ToucheSelectionnee* que *ChoixMenu* renvoie également.

ActiveElemMenu

Déclaration

```
procedure ActiveElemMenu(Mnu : Menu ; TcheM : ToucheMenu) ;
```

Objet

Permet à l'utilisateur de sélectionner l'élément de menu spécifié dans un menu donné.

Commentaires

Appelez cette procédure pour réactiver un élément de menu précédemment désactivé par *DesactiveElemMenu*. Voir cette routine pour plus de détails.

Vous pouvez activer plus d'une fois un élément de Menu.

Exemples

Voir les exemples de *DesactiveMenu*

Voir aussi

DesactiveElemMenu (TPMENU) InitToucheSelMenu (TPMENU)

ActiveMenuSouris

Déclaration

```
procedure ActiveMenuSouris;
```

Objet

Rend le contrôle des menu à la souris.

Commentaires

Appelez cette routine pour réactiver le contrôle d'un système de menus par la souris. La routine vérifie si la souris est installée, puis positionne un indicateur interne pour que les autres routines de menu utilisent la souris. Appelez une première fois *ActiveMenuSouris* avant d'appeler la routine CREEMENU de TPMENU.

Puisque le contrôle de la souris TPMENU dépend du gestionnaire d'événements clavier de TPSOURIS, *ActiveMenuSouris* appelle également la routine *ActiveAttenteEvenement* de TPSOURIS. N'ayez pas d'inquiétude si vous avez déjà appelé cette routine. N'oubliez pas cependant que tout gestionnaire d'événements déjà activé perdra le contrôle après l'appel à *ActiveMenuSouris*.

ActiveMenuSouris ne modifie pas la fenêtre courante de la souris, le type de curseur de la souris, ou le ratio Mickey-en-pixels, pas plus qu'elle n'affiche le curseur de la souris. Votre programme devra initialiser ces valeurs en appelant les routines appropriées de TPSOURIS. Vous pouvez utiliser un curseur souris programmé ou matériel.

Lorsque la souris est activée TPMENU exécute les actions suivantes pendant les choix de menu. Lorsque le bouton gauche de la souris est pressé, TPMENU vérifie si le curseur pointe sur un élément du sous-menu courant. Si oui, cet élément est sélectionné comme si la touche (ENTER) était pressée, sinon TPMENU vérifie si le pointeur est dirigé sur un élément dans un menu parent du sous menu courant. Si oui, et si le menu parent n'a pas été activé, TPMENU sélectionne l'élément parent. Le bouton droit de la souris est traité comme la touche (ESC). Lorsque les deux boutons sont pressés en même temps, TPMENU se comporte comme si la touche d'aide (F1) avait été pressée.

Les actions prises en compte lorsque les boutons de la souris sont pressés peuvent être configurés de la même façon que les touches. Voir l'appendice C pour plus de détails.

Remarquez que les actions de la souris prennent effet lorsque les boutons sont relâchés mais pas lorsqu'ils sont pressés. Cette technique permet d'utiliser de façon fiable des commandes qui exigent que les deux boutons soient pressés simultanément. Vous pouvez inverser ce mode de fonctionnement en

positionnant la constante *AttenteRelacheBouton* de TPSOURIS. Une fois cette modification effectuée, les actions de la souris sont prises en compte immédiatement après avoir pressé les boutons de la souris.

TPMENU détecte si le curseur de la souris est positionné sur un élément donné en comparant la position du curseur avec la zone mise en valeur lors de la sélection. Avec des programmes résidants utilisant TPMENU, vous ne pourrez pas activer la souris avant de lancer le programme résident. Vous devrez donc activer la souris après la mise en mémoire du programme résident et la désactiver avant le retour. D'autres considérations doivent être prises en compte lors de l'écriture du programme résident qui utilise la souris. Voir "TPSOURIS" au chapitre 2 pour plus de détails.

Exemple

```
program MenuDemo;
uses TpCrt, TpSouris, TpMenu;
var
   Ch : Char;
   Principal : Menu;
   Touche : ToucheMenu;
procedure InitialiseMenu(var M : Menu);
begin
   ...
end;
begin
   InitialiseMenu(Principal);
   ActiveMenuSouris;
   repeat
      Touche := ChoixMenu(Principal, Ch);
      GotoXY(1, 25);
      Write('Sélection = ',Touche, ' Code de la touche = ', Ord(Ch));
      Delay(1000);
      EffaceMenu(Principal, False);
   until (NumCmdMenu = TMSelecte) and (Touche = 6);
end.
```

Démontre la facilité de mise en oeuvre de la souris avec un système de menu. *InitMenu* est la même routine que celle décrite au début de cette partie. Avant d'appeler *ActiveMenuSouris*, vous pourrez insérer d'autres appels des routines de TPSOURIS tels que *CursSourisSoft*, *CursSourisHard* et *FenSouris*.

L'utilitaire CREEMENU utilise pleinement les possibilités de la souris qui sont offertes par TPMENU. Voir ce programme comme exemple réel.

Voir aussi

DesactiveMenuSouris (TPMENU) ActiveAttenteEvenement (TPSOURIS)

ChercheElemMenu

Déclaration

```
function ChercheElemMenu(Mnu : Menu; TcheM : ToucheMenu;
                         var Elem : ElemP): Boolean;
```

Objet

Recherche la touche TcheM associée à l'élément, renvoie "True" et son pointeur si elle est trouvée.

Commentaires

Cette fonction est destinée principalement aux usages internes. Etant donnée une touche d'élément, elle renvoie un pointeur sur l'enregistrement qui décrit en détails cet élément. Nous utilisons cette routine pour activer et désactiver des éléments et pour changer le nom associé à ces éléments. Vous pourrez trouver d'autres utilisations de cette fonction. Etudiez la déclaration *EnrElem* dans TPMENU.IN0 pour voir toutes les informations contenues dans un enregistrement d'élément.

Exemples

```
if ChercheElemMenu(Mnu, ToucheM, Element) then
   {La touche ToucheM est trouvée}
   with Element^ do begin
      WriteLn('Le nom de l''élément est ', string(Nom^));
      WriteLn('Elément activé : ',Autorise);
   end;
```

Recherche l'élément de menu associé à *TcheM,* et écrit le nom de l'élément et indique s'il est actuellement activé.

Voir aussi

DesactiveElemMenu (TPMENU) InitNomElemMenu (TPMENU)
LitNomElemMenu (TPMENU)

ChoixMenu

Déclaration

```
function ChoixMenu(Mnu : Menu;
                   var ToucheSelectionnee : Char): ToucheMenu;
```

Objet

Affiche un système de menus et permet à l'utilisateur de choisir une option.

Commentaires

ChoixMenu est la routine principale de TPMENU. Une application appellera très peu les autres routines, pour construire ou mettre en oeuvre un système de menus, au cours de l'exécution. Au contraire, elle appellera *ChoixMenu* de façon répétitive, lorsque l'utilisateur sélectionnera une option.

Le premier paramètre de *ChoixMenu* est un menu déjà initialisé par une instruction *NouvMenu* ou *LitBiblioMenu*. La fonction *SousMenu* affiche le sous-menu de niveau supérieur si le système de menu spécifié est déjà sur l'écran, sinon elle se poursuit après l'appel à *ChoixMenu*.

A l'intérieur de *ChoixMenu*, l'utilisateur peut utiliser un ensemble de touches prédéfinies pour se déplacer dans le système de menus. Les touches qui suivent sont les valeurs par défaut de TPMENU, mais peuvent être modifiées en utilisant les techniques décrites en appendice C. TPMENU supporte également les touches Wordstar.

(←)(→)(↑)(↓) : les flèches permettent de passer d'une option à une autre, à gauche, à droite, vers le haut, vers le bas et également de changer de sous-répertoire lorsque c'est logiquement nécessaire.

(ENTER) : s'il n'y a pas de sous-menu imbriqué attaché à l'option courante, la fonction *ChoixMenu* est interrompue et retourne la *ToucheMenu* de cette option ainsi que la touche utilisée pour la sélectionner ((ENTER) dans ce cas). Si un sous-menu est attaché, (ENTER) sélectionne le sous-menu.

(ESC) : si le sous-menu actuel est au niveau supérieur, la routine *ChoixMenu* sera interrompue et retournera la touche *Menu* de l'option en cours de sélection ainsi que la touche utilisée pour la sélectionner ((ESC) dans ce cas). Si le sous-menu courant est imbriqué, *ChoixMenu* l'effacera et retournera au sous-menu sélectionné précédemment.

Sélection par caractère : si vous le souhaitez, *ChoixMenu* permet de sélectionner les options lorsqu'un caractère l'identifiant est pressé. Généralement, le caractère sera la première lettre du nom de l'option. Lorsqu'une touche correspondante est pressée (les majuscules/minuscules sont

sans importance), *ChoixMenu* renvoie la touche *Menu* de l'option sélectionnée et renvoie la touche (ENTER) comme caractère de sélection.

(F1) : si la variable globale *PtrAideMenu* ne vaut pas nil, *ChoixMenu* appellera votre routine d'aide lorsque cette touche est pressée. Voir l'appendice C pour plus de détails.

L'application peut définir un ensemble supplémentaire de touches pour les options. Par exemple, si un utilisateur appuie sur la touche (F10), *ChoixMenu* pourra s'interrompre et renvoyer la *ToucheMenu* de l'option sélectionnée ainsi que la touche (F10). L'application pourra alors afficher une fenêtre, l'effacer et appeler *ChoixMenu* à nouveau pour laisser l'utilisateur terminer ses sélections.

L'ensemble de touches supplémentaires est défini au moment de l'appel de *NouvMenu* et peut être modifié à n'importe quel moment en appelant *InitToucheSelMenu*.

Au retour de *ChoixMenu*, tous les menus sélectionnés restent affichés. Des menus peuvent être effacés et l'écran recouvert restauré par un appel à *EffaceMenu*.

ChoixMenu renvoie deux éléments d'information, le résultat de la fonction et la *ToucheMenu*. Une *ToucheMenu* est de type *LongInt*. En pratique, la *ToucheMenu* peut contenir n'importe quel type de données inférieur ou égal à 4 octets. La *ToucheMenu* est un identificateur unique de chaque option dans un système de menus. Elle peut être un entier utilisé dans une instruction **case**, un type énuméré ou un pointeur sur une structure plus complexe contenant des détails supplémentaires. Une *ToucheMenu* est affectée à chaque option du menu pendant un appel à *MenuElem* lorsque le menu est initialisé.

Dans le paramètre VAR *ToucheSelectionnee*, *MenuChoix* renvoie la touche de l'option correspondante. Puisque ces données sont contenues dans un octet, TPMENU utilise une méthode de codage simple pour compacter des touches étendues en un seul octet. Les caractères normaux ASCII dans l'intervalle #0..#127 sont représentés normalement. Les touches étendues telles que (F1) (#0#59) sont représentées en éliminant le caractère nul et en positionnant le bit de poids fort du code de balayage. Ainsi, le code de (F1) est #187 (#59 avec le bit de poids fort mis à 1). Ce codage interdit l'utilisation du bit de poids fort normal (caractère semi-graphique ou étranger) des caractères pour la sélection mais ceci est peu probable.

Le code TPMENU de la commande utilisée pour sortir du système de menus est renvoyé dans la variable globale *NumCmdMenu*.

Exemple

```
repeat
   Touche := ChoixMenu(Principal, Ch);
   EffaceMenu(Principal, False);
   if NumCmdMenu = TMSelecte then
      case integer(Touche) of
         1 : {Exécute l'action correspondant à l'élément 1} ;
         2 : {Exécute l'action correspondant à l'élément 2} ;
         ...
      end;
until (NumCmdMenu = TMQuitte) or ((NumCmdMenu = TMSelecte)
      and (Touche = ToucheSortie));
```

Cet exemple boucle à l'infini tant qu'une option du menu n'est pas sélectionnée ou que la touche (ESC) n'est pas pressée au niveau supérieur. L'instruction Case sélectionne une valeur parmi les valeurs de *ToucheMenu* affectées à l'initialisation du menu.

Voir aussi

InitToucheSelMenu (TPMENU).

CompBiblioMenu

Déclaration

```
procedure CompBiblioMenu(NomE, NomS : string);
```

Objet

Supprime les entrées de menu superflues de la bibliothèque *NomE* et crée la bibliothèque *NomS*.

Commentaires

Cette procédure permet de gérer les bibliothèques de menus de TPMENU. Lorsqu'une nouvelle version de menus existants est écrite dans une bibliothèque de menus, la version précédente reste dans la bibliothèque mais elle est désactivée dans le répertoire de la bibliothèque. Si ce processus est répété, la bibliothèque de menus pourrait se saturer.

La procédure *CompBiblioMenu* lit une bibliothèque de menus existants (*NomE*) et écrit une version épurée de cette bibliothèque dans un nouveau fichier (*NomS*). Les systèmes de menus inutilisés sont désactivés et les entrées de répertoire inutilisées sont effacées. Le fichier de sortie ne doit pas avoir le même nom que le fichier d'entrée sinon la bibliothèque résultat sera endommagée. *CompBiblioMenu* ne fait pas de vérification.

CompBiblioMenu retournera une erreur si le fichier d'entrée n'est pas trouvé, si la bibliothèque n'a pas d'entrées désactivées ou si une erreur d'entrée/sortie se produit. *CompBiblioMenu* détruira le fichier de sortie sans avertissement ; les contrôles sont de votre responsabilité.

Voir *EcritBiblioMenu* pour plus de détails sur le format des bibliothèques de menus.

Exemple

```
CompBiblioMenu('MENU1.BIL , 'MENU2.BIB');
```

Compacte MENU1.BIB et crée une nouvelle bibliothèque MENU2.BIB.

Voir aussi

EtatMenu (TPMENU)
EcritBiblioMenu (TPMENU)
LitBiblioMenu (TPMENU)

ContrlEtatMenu

Déclaration

```
Procedure ContrlEtatMenu(EtatM : TypeEtatMenu);
```

Objet

Contrôle l'état d'un menu, s'arrête en cas d'erreur et renvoie son code.

Commentaires

ContrlEtatMenu offre un mécanisme simple (et quelquefois brutal) de gestion des erreurs pour les applications utilisant TPMENU. Si l'état d'un menu passé en paramètre indique une erreur, *ContrlEtatMenu* écrit un message et arrête le programme.

Cette procédure est pratique lors du développement d'applications utilisant TPMENU. Nous ne recommandons pas son utilisation dans des applications de production. Voir la fonction *EtatMenu* pour une gestion plus efficace des erreurs.

Exemple

```
ContrlEtatMenu(EtatMenu);
```

Passe l'état du menu actuel à *ContrlEtatMenu*. S'il n'y a pas d'erreur, *ContrlEtatMenu* revient simplement au programme appelant sinon elle écrit un message d'erreur et arrête le programme.

Voir aussi

EtatMenu (TPMENU)

DeallouеMenu

Déclaration

```
procedure DeallouеMenu(Mnu : Menu);
```

Objet

Déalloue tout l'espace réservé à un menu sur le tas.

Commentaires

TPMENU utilise le tas pour presque toutes ses structures de données. Lorsqu'un nouveau système de menus est initialisé, de la place est allouée en utilisant la procédure *GetMem*. Si une application se termine par un système de menus particulier, elle devra appeler *DeallouеMenu* qui utilise *FreeMem* pour restituer la place occupée sur le tas.

Exemple

```
DeallouеMenu(M);
```

Libère la place occupée sur le tas pour le menu *M*.

Voir aussi

ElemMenu (TPMENU)
SousMenu (TPMENU)
NouvMenu (TPMENU)

DesactiveElemMenu

Déclaration

```
procedure DesactiveElemMenu(Mnu : Menu ; Tche : ToucheMenu);
```

Objet

Permet à l'utilisateur de ne sélectionner que les éléments actifs dans un menu.

Commentaires

Il est très pratique de générer des systèmes de menus contextuels. Cela signifie que sous certaines conditions, un utilisateur d'un système de menu ne pourra pas sélectionner certains éléments du menu. L'utilitaire CREEMENU fournit un exemple : Lorsqu'aucun système de menu n'a été entré, CREEMENU sait qu'il n'y a rien à générer et le code source du menu sera sans signification. CREEMENU désactive alors cet élément du menu (et les autres), et les réactivera plus tard.

Lorsque vous désactivez un élément de menu avec *DesactiveElemMenu*, TPMENU fait deux choses : elle affiche cet élément avec une couleur différente (spécifiée en utilisant l'information *CouleurDesactivee* du tableau de couleur), et interdit ensuite toute sélection de l'élément par le déplacement du curseur. Ainsi les éléments restent affichés mais ne peuvent plus être sélectionnés. Notez que *DesactiveElemMenu* ne peut pas réafficher immédiatement le menu auquel appartiennent les éléments si bien que dans la plupart des cas, vous devrez appeler *RetraceMenu* après avoir désactivé un élément du menu.

Pour appeler *DesactiveElemMenu*, vous devez connaître la touche menu de l'élément. Cette touche est une valeur scalaire unique que vous spécifiez lorsque l'élément de menu est défini. Vous le trouverez en quatrième paramètre de *ElemMenu* ou *PtrElemMenu*.

Vous pouvez réactiver l'élément à n'importe quel moment en appelant *ActiveElemMenu* ; vous pouvez sans problème désactiver un élément plus d'une fois.

Si vous avez affecté une HotKey à un élément (comme le fait CREEMENU en utilisant les touches de contrôle), vous devrez le désactiver vous même en appelant *InitToucheSelMenu* après avoir modifié le jeu des caractères de HotKey. *DesactiveElemMenu* ne peut pas désactiver une HotKey à votre place.

Exemples

```
const
   EnsHotKey : EnsCaracMenu = [^R, ^V];
   TcheLectureFich = 1;
   TcheEcritureFich = 2;
var
   Mnu : Menu;
   SortiePrete : boolean;
begin
   {Définition du Menu}
   Mnu := NouvMenu(EnsHotKey, nil);
   ...
   ElemMenu('Lecture du fichier ', 1, 1, TcheLectureFich, '');
   ElemMenu('Ecriture du fichier ', 2, 1, TcheEcritureFich, '');
   ...
   if not SortiePrete then begin
      {Désactive la sélection de Menu pour l'écriture de fichier}
      DesactiveElemMenu(Mnu, TcheEcritureFich);
      {Désactive la hotkey pour l'écriture de fichier}
      EnsHotKey := EnsHotKey-[^W];
      InitToucheSelMenu(Mnu, EnsHotKey);
      {Reflète les modifications à l'écran}
      RetraceMenu(Mnu);
   end;
   ...
   if not SortiePrete then begin
      {Active la sélection de Menu pour l'écriture de fichier}
      ActiveElemMenu(Mnu, TcheEcritureFich);
      {Active la hotkey pour l'écriture de fichier}
      EnsHotKey := EnsHotKey+[^W];
      InitToucheSelMenu(Mnu, EnsHotKey);
      {Reflète les modifications à l'écran}
      RetraceMenu(Mnu);
   end;
end.
```

Cet exemple définit un système de menus fictif qui permet de sélectionner des opérations d'écriture et de lecture de fichiers. Il n'y a pas de données à écrire, et nous désactivons l'option d'écriture du fichier. Nous désactivons également la HotKey associée à l'écriture d'un fichier. Plus tard, lorsqu'un fichier sera prêt, nous réactiverons l'option d'écriture.

CREEMENU utilise ces possibilités pour lui-même. Voir CREEMENU.INC pour l'exemple code complet.

Voir aussi

ActiveElemMenu (TPMENU) InitToucheSelMenu (TPMENU)
RetraceMenu (TPMENU)

DesactiveMenuSouris

Déclaration

```
procedure DesactiveMenuSouris;
```

Objet

Désactive le contrôle du menu par la souris.

Commentaires

Cette routine annule les effets de *ActiveMenuSouris*. Dans de nombreux programmes, vous n'aurez pas besoin d'appeler *DesactiveMenuSouris* puisque ses fonctions principales sont gérées automatiquement par les routines Exit de *Turbo Pascal*. Dans certains cas, notamment avec les programmes résidents, vous pourrez appeler *DesactiveMenuSouris* afin de désactiver temporairement les fonctions de la souris.

Notez que *DesactiveMenuSouris* ne cache pas le curseur de la souris. Vous pouvez appeler *CacheSouris* pour cela.

Exemples

```
{Activation du TSR}
...
ActiveMenuSouris;
repeat
   Touche := ChoixMenu(Principal, Ch);
until (NumCmdMenu = TMSelecte) and (Touche = 6);
EffaceMenu(Principal, False);
DesactiveMenuSouris;
...
{Fin TSR}
```

Voir aussi

ActiveMenuSouris (TPMENU)

DessineMenuPile

Déclaration

```
Procedure DessineMenuPile(Mnu : Menu ; Var TPile : PileMenuP);
```

Objet

Dessine un système de menus en utilisant la pile des options sauvegardées précédemment.

Commentaires

Dans une application dirigée par les menus, vous aurez parfois à effacer l'ensemble des menus, mettre à jour l'écran et restaurer les menus. Deux procédures de TPMENU, *EffaceMenuPile* et *DessineMenuPile*, fournissent les moyens de faire cela.

Le second paramètre de *DessineMenuPile* est un pointeur sur la pile, initialisé précédemment par *EffaceMenuPile*.

Exemple

```
var
   M : Menu;
   PileM : PileMenuP;
begin
   {Suppression des menus en cours}
   EffaceMenuPile(M, PileM);
   {L'écran est restauré, mise à jour de l'écran}
   ...
   {Redessine les menus choisis}
   DessineMenuPile(M, PileM);
end.
```

Voir aussi

EffaceMenuPile (TPMENU)

EcritBiblioMenu

Déclaration

```
procedure EcritBiblioMenu(Mnu : menu; NomF : string; ID : string);
```

Objet

Ecrit un système de menus sur une bibliothèque en format binaire.

Commentaires

TPMENU définit un format de bibliothèque pour les menus. Chaque bibliothèque peut contenir au plus dix systèmes de menus. Pour écrire dans une bibliothèque de menus, vous devez définir un nom de fichier et un identificateur pour chaque système de menus à l'intérieur de la bibliothèque. L'identificateur de menu *ID* est une chaîne de 12 caractères utilisée pour différencier les menus à l'intérieur d'une bibliothèque. *EcritBiblioMenu* écrit un menu défini précédemment en mémoire sur un fichier de menus. Le menu en mémoire est spécifié par le paramètre *Mnu*. Le nom du fichier est *NomF* et l'identificateur de menu est *ID*.

Si le fichier dont le nom est dans *NomF* n'existe pas, *EcritBiblioMenu* le créera. Si le fichier existe mais que ce n'est pas un fichier de TPMENU, *EcritBiblioMenu* se terminera par une erreur. Si l'identificateur de *Menu* spécifie un menu existant dans la bibliothèque, *EcritBiblioMenu* marquera la version existante du menu comme superflue et ajoutera la nouvelle version dans la bibliothèque. *EcritBiblioMenu* désactive alors l'entrée du répertoire de la bibliothèque de menus mais laisse le système de menus intact à l'intérieur de la bibliothèque. Les menus superflus sont potentiellement récupérables et continuent d'occuper de l'espace disque. La procédure *CompBiblioMenu* permet d'effacer les menus superflus dans une bibliothèque.

EcritBiblioMenu écrit une description complète du système de menus spécifié mais ne permet pas de définir une fonction d'affichage utilisateur. L'adresse d'une telle fonction pouvant être différente lors de la prochaine lecture d'un menu dans une bibliothèque, l'enregistrement de cette information sera sans signification. La fonction *LitBiblioMenu* accepte la spécification d'une fonction d'affichage utilisateur quand un menu est lu dans une bibliothèque.

Le système de menus reste intact en mémoire même lorqu'il a été enregistré sur disque. Appelez *DeallouéMenu* si vous voulez l'effacer de la mémoire. *EcritBiblioMenu* renvoie une erreur en cas de problèmes disque. Voir la partie déclaration de *TypeEtatMenu* pour plus de détails sur les erreurs possibles.

Exemple

```
EcritBiblioMenu(Mnu, 'Menu.BIB', 'MENU1');
```

Ecrit le système de menus *Mnu* dans le fichier MENU.BIB avec l'identificateur MENU1.

Voir aussi

LitBiblioMenu (TPMENU)

EffaceMenu

Déclaration

```
procedure EffaceMenu(Mnu : Menu; InitSelections : Boolean);
```

Objet

Efface les menus actifs de l'écran et réinitialise les valeurs de sélection si nécessaire.

Commentaires

EffaceMenu efface les menus affichés et restaure l'écran recouvert. Si *InitSelections* a la valeur "True", la sélection courante de tous les sous-menus est réinitialisée avec la première option de chacun d'eux. Sinon, toutes les sélections gardent leurs valeurs courantes.

Après avoir utilisé *EffaceMenu*, lorsqu'une application appellera *ChoixMenu*, elle affichera le niveau primaire du sous-menu. Si vous voulez que *ChoixMenu* revienne à un menu intermédiaire, vous devrez utiliser la procédure *EffaceMenuPile* au lieu de *EffaceMenu*. *EffaceMenuPile* enregistre les informations sur les sous-menus sélectionnés pendant l'effacement. *DessineMenuPile* restaurera alors le système de menus.

Exemple

```
EffaceMenu(M,True);
```

Efface les menus de l'écran et réinitialise toutes les sélections.

Voir aussi

EffaceMenuPile (TPMENU) ChoixMenu (TPMENU)

EffaceMenuPile

Déclaration

```
procedure EffaceMenuPile(Mnu : Menu; var TPile : PileMenuP);
```

Objet

Efface un système de menus en sauvegardant le chemin de l'option actuelle sur la pile.

Commentaires

Dans une application avec menus, il peut arriver que vous souhaitiez effacer l'ensemble actuel de menus, mettre à jour l'écran recouvert et ensuite, restaurer les menus à leur état précédent. Les deux procédures de TPMENU, *EffaceMenuPile* et *DessineMenuPile*, fournissent le moyen de faire cela.

Le deuxième paramètre de *EffaceMenuPile* est un pointeur qui contiendra le sommet de la pile des sous-menus sélectionnés. *EffaceMenuPile* efface les sous-menus de l'écran, restaure l'écran recouvert et construit une pile que *DessineMenuPile* peut utiliser pour réafficher le système de menus.

Exemple

```
var
   M : Menu;
   PileM : PileMenuP;
begin
   {Suppression des menus en cours}
   EffaceMenuPile(M, PileM);
   {L'écran est restauré, mise à jour de l'écran}
   ...
   {Redessine les menus choisis}
   DessineMenuPile(M, PileM);
end.
```

Voir aussi

DessineMenuPile (TPMENU)

EffaceSousMenuActuel

Déclaration

```
procedure EffaceSousMenuActuel(Mnu : Menu);
```

Objet

Efface le sous menu actuel affiché.

Commentaires

Lorsque l'utilisateur a sélectionné une option avec *ChoixMenu*, le système de menus reste affiché. Votre programme peut alors mettre à jour l'écran de plusieurs manières :

- Laisser l'écran inchangé et revenir ultérieurement au même élément de menu en appelant *ChoixMenu* à nouveau.
- Effacer tout le système de menus avec *EffaceMenu* et entrer ultérieurement dans le système de menus au niveau primaire avec *ChoixMenu*.
- Effacer tout le système de menus avec *EffaceMenu* et activer ultérieurement le système de menus à n'importe quel niveau en appelant *SelectElemMenu* avant d'appeler *ChoixMenu*.
- Effacer le sous menu courant avec *EffaceSousMenuActuel* et activer ultérieurement le système de menus parent de l'élément déjà sélectionné avec *ChoixMenu*.

Mais vous appelerez *EffaceSousMenuActuel* lorsque vous désirerez effacer le dernier sous-menu sélectionné sur l'écran. Si des sous-menus imbriqués restent affichés, vous pouvez appeler *EffaceSousMenuActuel* de façon répétitive.

Exemples

```
repeat
   ToucheM := ChoixMenu(Mnu, CH);
   if NumCmdMenu = TMSelecte then
      case ToucheM of
         1 : {option d'effacement du sous-Menu courant}
            begin
               {Exécute les actions inhérentes à ce choix}
               {Efface le sous-Menu avant de lire le choix de Menu
                suivant}
               EffaceSousMenuActuel(Mnu);
            end;
         2 : {Exécute les actions inhérentes à ce choix} ;
      end;
until Fin;
```

Avec ce système de menus, l'élément 1 exécute une action que nous ne voulons pas répéter immédiatement. Par conséquent nous effaçons son sous-menu avant de boucler pour lire le choix de menu suivant.

```
with Mnu^ do
   while Visible and (Actif <> Racine) do
      EffaceSousMenuActuel(Mnu);
```

Cette boucle efface tout le sous-menu du menu *Mnu* jusqu'à ce que seul le menu primaire reste affiché.

Voir aussi

EffaceMenu (TPMENU)

ElemMenu

Déclaration

```
procedure ElemMenu(NomChaine : string; AffichePosP : Byte;
                   SelectPosP : Byte; ToucheP : ToucheMenu
                   ChaineAide : string);
```

Objet

Ajoute une option dans un sous-menu.

Commentaires

Appelez cette procédure pendant la création d'un système de menus. Après avoir défini au préalable un sous-menu (en appelant *SousMenu*), appelez *ElemMenu* pour chaque option du sous-menu.

Chaque appel de *ElemMenu* définit le nom de l'option (*NomChaine*), l'offset de l'option à partir du coin supérieur gauche du sous-menu (*AffichePosP*), l'offset à l'intérieur du nom du caractère de sélection (*SelectPosP*), la *ToucheMenu* renvoyée lorsque l'option a été sélectionnée (*ToucheP*) et une chaîne d'aide facultative à afficher pendant que le curseur est sur l'option (*ChaineAide*).

La signification de *AffichePosP* est fonction de l'orientation du sous-menu qui peut être horizontale ou verticale. Si l'orientation est horizontale, *AffichePosP* est égale à l'offset de la colonne à partir du bord gauche du sous-menu, si l'orientation est verticale, *AffichePosP* est égale à l'offset de la ligne à partir du bord gauche du sous-menu.

Si *SelectPosP* vaut 1, le premier caractère du nom de l'option sera affiché avec un attribut *CouleurSurbrillance*. Ce caractère sert également à sélectionner une option du menu. Si *SelectPosP* vaut zéro, aucun caractère ne sera mis en valeur ou utilisé dans la sélection.

ToucheP est de type *ToucheMenu* mais peut être de n'importe quel type de données de 4 octets ou moins. Vous pouvez définir d'autres types pour *ToucheMenu* lors de l'appel à *ElemMenu*. Nous vous recommandons de choisir les valeurs des touches de menu dans l'intervalle 1 à 10 000, afin d'assurer la compatibilité avec le système d'aide contextuel TPAIDE.

ElemMenu alloue de la place sur le tas pour enregistrer *NomChaine* et *ChaineAide*. Si ces chaînes de caractères sont déjà enregistrées comme des données statiques, vous devrez appeler *PtrElemMenu* au lieu de *ElemMenu*. *PtrElemMenu* n'alloue pas de place supplémentaire pour les chaînes de caractères. *ElemMenu* peut renvoyer une erreur s'il n'y a pas assez de place pour enregistrer les chaînes de caractères.

ElemMenu insère la nouvelle option dans le sous-menu actif immédiatement après l'option active et fait de cette nouvelle option l'option actuelle. En utilisant l'utilitaire CREEMENU pour concevoir un système de menus interactivement, vous n'aurez pas à vous tracasser avec les nombreux détails de la procédure *ElemMenu*.

Exemple

```
SousMenu(1, 2, 25, Horizontal, Cadre1, Couleur1, 'TEST');
ElemMenu('Fichier', 2, 1, ToucheMenu(OpsFich),
         'Aide pour opérations sur fichiers');
```

Ajoute le premier élément du menu au sous-menu horizontal appelé "TEST". Le nouvel élément est appelé "Fichier", et il se trouve à deux colonnes à droite du coin du sous-menu, la lettre "F" est le caractère de selection, il retournera *ToucheMenu* dont la valeur est "OpsFich" et affichera une ligne d'aide en ligne 25 lorsqu'il sera sélectionné.

Voir aussi

PtrElemMenu (TPMENU)

SousMenu (TPMENU)

EtatMenu

Déclaration

```
function EtatMenu: TypeEtatMenu;
```

Objet

Renvoie l' état de l'opération précédente.

Commentaires

EtatMenu fonctionne comme la fonction *IoResult* de Turbo. *EtatMenu* renvoie l'erreur actuelle et réinitialise ensuite la variable erreur interne. Voir la partie "déclaration", *TypeEtatMenu,* pour les différentes valeurs d'erreurs.

Exemple

```
EtatM := EtatMenu;
if EtatM <> SuccesMenu then begin
   WriteLn('Erreur ', Ord(EtatM), ' pendant une opération lancée
                depuis un Menu');
   Halt(1);
end;
```

Voir aussi

ContrlEtatMenu (TPMENU) Déclarations (TPMENU)

HauteurMenu

Déclaration

```
procédure HauteurMenu(Hauteur : Byte);
```

Objet

Initialise la hauteur du sous-menu en cours de construction.

Commentaires

TPMENU adapte automatiquement la taille minimum d'un sous-menu qui encadrera les différents éléments d'un menu. Avec *LargeurMenu*, *HauteurMenu* vous permet d'agrandir un menu si vous voulez utiliser des menus de style Lotus de la même taille, des systèmes de menus déroulants qui occupent tout l'écran, un menu comportant une zone dans la partie inférieure pour une fenêtre popup séparée, ou un autre menu plus grand mieux adapté à votre application.

Appelez *HauteurMenu* tant qu'un sous-menu particulier est en cours de construction. Appelez *HauteurMenu* après l'appel de *SousMenu* qui crée le sous-menu (le code source généré par CREEMENU place toujours les appels à *HauteurMenu* et *LargeurMenu* de cette façon). Si vous le souhaitez, vous pouvez appeler *HauteurMenu* après l'appel à *SousMenu* et avant le prochain appel à *LanceSousNiveau*.

Le paramètre *Hauteur* spécifie le nombre total de lignes d'un menu y compris le cadre s'il existe. Si la hauteur est inférieure au nombre minimum de lignes nécessaires pour afficher tous les éléments, elle sera ignorée et la valeur prise par défaut sera la plus grande hauteur.

Exemple

```
M := NouvMenu([], Nil);
SousMenu(1,2,1, Horizontal, Cadre1, Couleur1, 'TPMENU');
   LargeurMenu(30);
   ElemMenu('Premier',2,1,1, '');
   SousMenu(2,4,1,Vertical,Cadre2,Couleur1,'');
      LargeurMenu(20);
      HauteurMenu(6);
      ModeMenu(True, True, False);
      ElemMenu('Alpha',1,1,2,'');
      ElemMenu('Beta',2,1,3,'');
      LanceSousNiveau;
   ElemMenu('Deuxième',12,1,4,'');
```

```
SousMenu(11,4,1,Vertical,Cadre2,Couleur1,'');
   LargeurMenu(20);
   HauteurMenu(6);
   ModeMenu(True, True, False);
   ElemMenu('Gamma',1,1,5,'');
   ElemMenu('Delta',2,1,6,'');
   LanceSousNiveau;
LanceSousNiveau;
InitMenu(M);
```

Ce programme reprend l'exemple du début et ajoute des appels à *HauteurMenu* et *LargeurMenu*. Le menu horizontal supérieur comporte 30 colonnes et a donc une taille supérieure à la valeur par défaut de 17. Le sous-menu vertical comporte 20 colonnes et 6 lignes. Ces sous-menus utilisent des cadres qui explosent avec des ombres.

Voir aussi

ModeMenu (TPMENU)

LargeurMenu (TPMENU)

InitMenu

Déclaration

```
procedure InitMenu(Mnu : Menu);
```

Objet

Initialise toutes les sélections avec la première option.

Commentaires

Après la construction d'un système de menus, toutes les sélections sont positionnées par défaut sur la dernière option de chaque sous-menu. *InitMenu* réinitialise toutes les sélections avec la première option de chaque sous-menu.

InitMenu est parfois utile après l'appel à *ChoixMenu*. Sinon, TPMENU mémorise les sélections précédentes tant qu'elles ne sont pas modifiées lors des appels à *ChoixMenu*.

Exemple

```
InitMenu(Mnu);
```

Réinitialise toutes les sélections du système de menu *Mnu*.

InitMenuDelai

Déclaration

```
procedure InitMenuDelai(Mnu : Menu; DelaiExplosion : Word);
```

Objet

Modifie le délai séparant deux explosions pour tout le système de menus.

Commentaires

Cette routine modifie le délai séparant les explosions de fenêtres. Si aucun sous-menu du système de menus ne comporte d'explosions actives, l'appel à *InitMenuDelai* est sans effet.

DelaiExplosion est exprimé en millisecondes et représente le délai séparant deux explosions successives d'une fenêtre.

Bien que le délai séparant deux explosions soit une information globale du système de menus, chaque sous-menu peut individuellement préciser une explosion ou non. Toutes les explosions d'un système de menus peuvent être temporairement désactivées en positionnant un délai d'explosion à 0.

Il n'est pas obligatoire que le système de menus soit affiché ou non lorsque *InitMenuDelai* est appelée.

Exemple

```
var
   Mnu : Menu;
   SauveDelai : Word;
begin
   {Construction du système de menus}
   ...
   {Sauvegarde du délai d'explosion}
   SauveDelai := Mnu^.Racine^.Dessine.DureeExplosion;
   {Désactive les explosions}
   InitMenuDelai(Mnu, 0);
   {Réactive les explosions}
   InitMenuDelai(Mnu, SauveDelai);
end.
```

Cette exemple montre comment désactiver temporairement les explosions d'un système de menus.

Voir aussi

ModeMenu (TPMENU)

InitModeMenu (TPMENU)

InitModeMenu

Déclaration

```
procedure InitModeMenu(Mnu : Menu; Explosion, Ombre, Bruit : Boolean);
```

Objet

Modifie le mode d'affichage de tout un système de menus.

Commentaires

InitModeMenu modifie l'affichage de tous les sous-menus d'un système de menus. Après l'appel de cette routine, tous les sous-menus utiliseront ou non des explosions, des ombres ou des bruitages. Les sons apparaîtront seulement pendant l'explosion d'un sous-menu.

Il n'est pas nécessaire que le système de menus soit affiché lorsque *InitModeMenu* est appelée. Si les ombres sont activées lorsque *InitModeMenu* est appelée, elles n'apparaîtront pas tant que le système de menus n'est pas redessiné.

Exemple

```
var
   Mnu : Menu;
begin
   {Construction du système de menus}
   InitModeMenu(Mnu, False, True, False);
   {Redessine le système de menus afin que les ombres soient visibles}
   RetraceMenu(Mnu);
end.
```

Voir aussi

ModeMenu (TPMENU)

InitMenuDelai (TPMENU)

InitNomElemMenu

Déclaration

```
function InitNomElemMenu(Mnu : Menu;
                         TcheM : ToucheMenu;
                         NouvNom : string): Boolean;
```

Objet

Modifie la chaîne décrivant un élément de menu donné.

Commentaires

Cette fonction vous permet de renommer dynamiquement les éléments de menus. Spécifiez la touche de l'élément TcheM pour l'élément que vous désirez renommer et donnez le nouveau nom de l'élément dans le paramètre *NouvNom*.

InitNomElemMenu recherche tout d'abord le système de menus pour la touche TcheM. Si la touche n'existe pas, *InitNomElemMenu* renverra la valeur "False".

Les deux noms doivent être exactement de la même longueur sinon le nouveau nom sera tronqué ou complété par des blancs.

Après l'appel à *InitNomElemMenu*, vous pouvez appeler *RetraceMenu* pour afficher les modifications à l'écran.

Exemple

```
const
   TcheEcritureFich = 2;
var
   Mnu : Menu;
   ToucheM : ToucheMenu;
   Ch : Char;
begin
   {Définition du Menu}
   Mnu := NouvMenu([], Nil);
   ...
   ElemMenu('Ecriture fichier', 2, 1, TcheEcritureFich, '');
   ...
   {Lecture d'un choix. Le système de menus reste à l'écran}
   ToucheM := ChoixMenu(Mnu, Ch);
   ...
```

```
    {Renomme l'élément d'écriture de fichier}
    if InitNomElemMenu(Mnu, TcheEcritureFich, 'Création de fichier')
       then
       {Reflète les modifications à l'écran}
       RetraceMenu(Mnu);
end.
```

Sous certaines conditions, ce programme renomme l'élément de menu 'Ecriture fichier' en 'Création de fichier' et met à jour l'écran pour afficher les modifications. Le nouveau nom étant plus long que l'ancien nom, il sera tronqué. Pour éviter cela, vous pouvez compléter l'ancien nom par un ou plusieurs blancs.

Voir aussi

LitNomElemMenu (TPMENU) RetraceMenu (TPMENU)

InitToucheSelMenu

Déclaration

```
procedure InitToucheSelMenu(Mnu : Menu; TouchesS : EnsCaracMenu);
```

Objet

Modifie l'ensemble des touches de sélection d'un système de menus existant.

Commentaires

Avec cette routine, vous pouvez modifier dynamiquement l'ensemble des touches de sélection d'un menu. Voir la description de *NouvMenu* pour une description complète des touches de sélection complémentaires.

Pourquoi voudriez-vous modifier les touches de sélection ? Peut-être parce que vous voulez désactiver la touche d'aide pendant certaines opérations. Peut-être enregistrez-vous des menus dans une bibliothèque (avec un nombre déterminé de touches de sélection) et voulez-vous personnaliser certaines "hotkey" ?

Exemple

```
InitToucheSelMenu(Mnu,[]);
```

Désactive toutes les touches de sélection complémentaires.

```
InitToucheSelMenu(Mnu,[#187]);
```

Affecte (F1) comme seule touche de sélection complémentaire.

Voir aussi

NouvMenu (TPMENU)

LanceSousNiveau

Déclaration

```
procedure  LanceSousNiveau;
```

Objet

Lance le menu actif situé au sommet de la pile de menus.

Commentaires

Appelez *LanceSousNiveau* pendant l'initialisation d'un système de menus à plusieurs niveaux. *LanceSousNiveau* indique à TPMENU que le sous-menu actuel est terminé et que l'initialisation devrait se poursuivre avec l'option suivante dans le menu parent.

Remarquez que le processus d'initialisation d'un menu se fait en profondeur d'abord. Si une option de menu possède un sous-menu, le sous-menu doit être défini entièrement avant de continuer avec l'option suivante du menu. En utilisant le programme CREEMENU, vous éviterez les tracasseries relatives aux détails de la définition des menus.

Exemples

Voir l'exemple de l'introduction de cette partie pour l'utilisation de *LanceSousNiveau* dans un système de menus à deux niveaux.

Voir aussi

SousMenu (TPMENU)

LargeurMenu

Déclaration

```
procedure LargeurMenu(Largeur : Byte);
```

Objet

Initialise la largeur d'un sous-menu en cours de construction.

Commentaires

TPMENU adapte automatiquement la taille minimum d'un sous-menu qui encadrera les différents éléments d'un menu. *LargeurMenu*, avec *HauteurMenu*, vous permet d'agrandir un menu si vous voulez utiliser des menus de style Lotus de la même taille, des systèmes de menus déroulants qui occupent tout l'écran, un menu comportant une zone dans la partie inférieure pour une fenêtre popup séparée, ou un autre menu plus grand mieux adapté.

Appelez *LargeurMenu* tant qu'un sous-menu particulier est en cours de construction. Appelez *LargeurMenu* immédiatement après l'appel de *SousMenu* qui crée le sous-menu (le code source généré par CREEMENU place toujours les appels à *HauteurMenu* et *LargeurMenu* de cette façon). Si vous le souhaitez, vous pouvez appeler *LargeurMenu* après l'appel à *SousMenu* et avant le prochain appel à *LanceSousNiveau*.

Le paramètre *Largeur* spécifie le nombre total de colonnes d'un menu y compris le cadre s'il existe. Si la largeur est inférieure au nombre minimum de colonnes nécessaires pour afficher tous les éléments, elle sera ignorée et la valeur prise sera la plus grande largeur.

Exemple

Voir l'exemple de la procedure *HauteurMenu*.

Voir aussi

HauteurMenu (TPMENU)

LitBiblioMenu

Déclaration

```
function LitBiblioMenu(NomF : string; ID : string;
                       UtlFonc : Pointer):Menu;
```

Objet

Lit un système de menus à partir d'une bibliothèque de menus en format binaire.

Commentaires

TPMENU définit un format de bibliothèque pour les menus. Chaque bibliothèque peut contenir au plus dix systèmes de menus. Pour accéder à un système de menus défini, vous devez connaître le nom du fichier bibliothèque et également l'identificateur *ID* du menu. Le paramètre *ID* est une chaîne de longueur égale à 12, utilisée pour différencier les différents menus à l'intérieur d'un fichier. Pour plus de détails sur la création de bibliothèque de menus, voir la procédure *EcritBibliomenu*.

LitBiblioMenu traite le nom du fichier et l'identificateur de menu *ID* comme des paramètres. Elle prend aussi comme paramètre un pointeur sur une fonction utilisateur qui personnalise la présentation des options sur le menu. Si cette caractéristique n'est pas désirée, passez la valeur nil dans le paramètre *UtlFonc*. Voir la description de *NouvMenu* pour plus de détails sur *UtlFonc*.

LitBiblioMenu renvoie un pointeur sur un système de menus complètement initialisé. Elle renverra une erreur si le fichier ou le menu spécifié n'est pas trouvé ou s'il n'y a pas assez de place sur le tas pour enregistrer le système de menus en mémoire.

Exemple

```
Mnu := LitBiblioMenu('Menu.BIB', 'MENU1', Nil);
if EtatMenu <> SuccesMenu then begin
   WriteLn('Erreur pendant la lecture du Menu');
   Halt;
end;
```

Lit le menu 'MENU1' sur le fichier bibliothèque 'MENU.BIB'. Aucune fonction d'affichage n'est spécifiée. Un contrôle d'erreur garantit que le menu sera lu correctement.

Voir aussi

NouvMenu (TPMENU)

EcritBiblioMenu (TPMENU)

LitNomElemMenu

Déclaration

```
function LitNomElemMenu(Mnu: Menu; TcheM : ToucheMenu;
                        var Nom : string): Boolean;
```

Objet

Renvoie la chaîne associée à l'élément de menu spécifié. La valeur de la fonction est "True" si l'élément existe et "False" sinon.

Commentaires

Cette routine recherche dans le système de menus une touche de valeur *TcheM*. Si l'élément n'existe pas, *LitNomElemMenu* renvoie "False" et une chaîne vide. Sinon, elle renvoie le nom associé à cette élément.

Cette fonction peut être très utile pour construire des messages d'erreurs personnalisés basés sur des sélections de menu.

Exemple

```
const
   TcheLectureFich = 1;
   TcheEcritureFich = 2;
var
   Mnu : Menu;
   Nom : string;
begin
   {Définition du Menu}
   Mnu := NouvMenu([], Nil);
   ...
   {Eléments du Menu}
   ElemMenu('Lecture fichier', 1, 1, TcheLectureFich, '');
   ElemMenu('Ecriture fichier', 2, 1, TcheEcritureFich, '');
   ...
   {Retourne le nom d'un élément}
   if LitNomElemMenu(Mnu, TcheLectureFich,Nom) then
      WriteLn('Le nom de l''élément associé à TcheLectureFich est : ',
               Nom)
   else
      WriteLn('Aucun élément assoscié à TcheLectureFich');
end.
```

Ce programme fournit le nom associé à *TcheLectureFich* qui est 'Lecture fichier'.

Voir aussi

InitNomElemMenu (TPMENU)

ModeMenu

Déclaration

```
procedure ModeMenu(Explosion, Ombre, Bruit : Boolean);
```

Objet

Initialise le mode du sous-menu en cours de construction.

Commentaires

TPMENU fait appel à l'unité TPFENETR pour dessiner les cadres de chaque sous-menu. Par conséquent, TPMENU offre les mêmes possibilités que TPFENETR et permet des effets spéciaux : explosion de cadres, fenêtres ombrées et effets sonores pendant les explosions. Par défaut, TPMENU n'utilise aucun des ces modes et produit des sous-menus sans ombre, sans explosion et sans bruit.

ModeMenu est utilisé pour modifier le mode d'affichage par défaut d'un sous-menu. Elle doit être appelée pendant la construction d'un sous-menu et doit être immédiatement suivie d'un appel à *SousMenu* pour définir le sous-menu. Ainsi chaque sous-menu d'un système de menus peut comporter différents modes d'affichage tels que ombre et explosion en fonction de vos goûts.

Le temps séparant deux explosions de fenêtre est une information globale au système de menus. Utilisez *InitMenuDelai* pour modifier cette valeur.

Exemple

Voir le code l'exemple de la procédure *HauteurMenu*.

Voir aussi

HauteurMenu (TPMENU)
LargeurMenu (TPMENU)
InitModeMenu (TPMENU)
InitMenuDelai (TPMENU)

NouvMenu

Déclaration

```
function NouvMenu(TouchesSel : EnsCaracMenu;
                  UtlFonc : Pointer) : Menu;
```

Objet

Initialise un nouveau système de menus.

Commentaires

Appelez *NouvMenu* lorsque vous voulez initialiser un nouveau système de menus. *NouvMenu* alloue un petit bloc de contrôle sur le tas et renvoie un pointeur sur ce bloc. Le pointeur renvoyé est utilisé comme identificateur pour toutes les opérations ultérieures sur les menus.

Les paramètres de *NouvMenu* définissent deux propriétés globales d'un système de menus. *TouchesSel* est un ensemble de caractères pour sélectionner les options de menu différemment. Si *TouchesSel* est un ensemble vide, seule la touche (ENTER) permettra à l'utilisateur de sélectionner une option sur le menu. En ajoutant des caractères à *TouchesSel,* vous permettez à l'utilisateur de sélectionner une option en appuyant sur (F1) ou grâce à une combinaison de touches "hotkey" permettant d'accéder rapidement a toute partie du système de menus.

Puisque les ensembles sont limités à un octet, TPMENU utilise une méthode de codage simple pour compacter des touches fonctions étendues sur un seul octet. Les caractères normaux ASCII dans l'intervalle #0..#127 sont représentés normalement. Les touches fonctions étendues telles que (F1) (#0#59) sont représentées en éliminant le caractère nul et en mettant à 1 le bit de poids fort du code de balayage. Ainsi, le code de (F1) est #187 (#59 avec le bit de poids fort à 1). Ce codage interdit l'utilisation du bit de poids fort (caractère semi-grahique ou étranger) pour la sélection mais cette utilisation est peu probable.

Le paramètre *UtlFonc* est une autre caractéristique puissante du système TPMENU. *UtlFonc* est un pointeur sur une fonction fournie par votre application. Cette fonction peut être appelée pour chaque option affichée sur l'écran. Le rôle de cette fonction est de personnaliser la chaîne de caractères affichée pour l'option en lui asssociant des informations complémentaires. Par exemple, considérons une option de menu utilisée pour passer du mode insertion au mode recouvrement lors de l'utilisation d'un traitement de texte. Il est souhaitable que l'option elle-même affiche l'état actuel de ce mode. Ceci peut être réalisé grâce à la fonction *UtlFonc*.

Si *UtlFonc* vaut nil, aucune fonction utilisateur n'est définie et cette caractéristique de TPMENU sera inutilisée. Sinon, *UtlFonc* doit contenir l'adresse d'une fonction avec la déclaration suivante :

```
{$F+}
function FoncAffiche(S : string; T : ToucheMenu) : string;
   {Personnalise la chaîne affichée pour chaque élément du Menu}
begin
end;
{$F-}
```

Remarquez que la fonction doit être compilée avec l'option FAR. Si vous oubliez l'option de compilation {$F+}, votre programme s'arrêtra brutalement. La fonction possède en entrée les deux paramètres ci-dessus et renvoie en résultat une chaîne de caractères. Le nom de la fonction est arbitraire.

En entrant dans la fonction utilisateur, la chaîne *S* doit contenir une version non personnalisée de la chaîne de caractères qui doit être affichée pour une option particulière du menu. La chaîne de caractères doit avoir été étendue à la longueur finale et le nom de l'option sera justifié à gauche à l'intérieur de la chaîne. Le paramètre *K* contiendra la *ToucheMenu* unique qui identifie chaque option dans le système de menus. La fonction utilisateur devra exécuter une sélection en fonction de cette touche et personnaliser la chaîne S comme désirée. Avant le retour, la fonction doit affecter la chaîne personnalisée au résultat de la fonction. Si aucune personnalisation n'est désirée pour une *ToucheMenu* particulière, la chaîne S devra être directement affectée au résultat de la fonction. La chaîne de caractères résultante doit toujours avoir une longueur égale à la chaîne d'entrée sinon l'écran sera probablement embrouillé.

Exemples

```
Mnu := NouvMenu([], Nil);
```

C'est le moyen le plus simple d'appeler *NouvMenu*. Aucune touche de sélection supplémentaire n'est définie et aucune fonction d'affichage n'est activée. Le système de menu sera repéré par la variable *Mnu*.

```
Mnu := NouvMenu([#187, ^A], @FoncAffiche);
```

Les touches (F1) et (CTRL)-A sont activées comme des touches de sélection complémentaires et *FoncAffiche* devient la fonction d'affichage.

Voir aussi

InitToucheSelMenu (TPMENU)

PtrElemMenu

Déclaration

```
procedure PtrElemMenu(NomPtr : Pointeur; AffichePosP : Byte;
                      SelectPosP : Byte; ToucheP : ToucheMenu;
                      PtrAide : Pointeur);
```

Objet

Ajoute une option dans un sous-menu.

Commentaires

PtrElemMenu fonctionne comme *ElemMenu* avec une exception. *PtrElemMenu* n'alloue pas de place sur le tas pour enregistrer les chaînes *Nom* et *Aide* de l'option. L'application appelante doit fournir des pointeurs sur ces chaînes qui doivent déjà se trouver à des adresses mémoire statiques.

PtrElemMenu est utile si les chaînes de caractères utilisées par un menu sont définies comme des constantes typées ou si elles sont lues à partir d'un fichier et enregistrées dans des variables globales ou sur le tas. N'utilisez pas *PtrElemMenu* si des chaînes de caractères du menu sont enregistrées dans des variables locales.

Généralement, *PtrElemMenu* est plus efficace pour la gestion de la mémoire que *ElemMenu*. Les chaînes de caractères littérales passées à *ElemMenu* sont enregistrées une première fois dans le segment de code par le compilateur et une autre fois sur le tas par *ElemMenu*. Au contraire, les chaînes de caractères des constantes typées utilisées avec *PtrElemMenu* sont enregistrées une seule fois dans le segment de données. L'utilisation de *PtrElemMenu* permet d'éviter cette double allocation et peut acroître l'efficacité de l'enregistrement des systèmes de menus importants. Voir *ElemMenu* pour plus de détails sur les autres aspects de *PtrElemMenu*.

Exemple

```
const Nom1 : string[7] = 'Fichier';
   Aide1 : string[33] = 'Aide pour opérations sur fichiers';
begin
   ...
   PtrElemMenu(@Nom1, 2, 1, ToucheMenu(OpsFich), @Aide1);
   ...
end;
```

L'opérateur '@' Turbo passe un pointeur à la constante chaîne de caractères typée correspondant dans *PtrElemMenu*.

Voir aussi

ElemMenu (TPMENU)

RetraceMenu

Déclaration

```
procedure RetraceMenu(Mnu : Menu);
```

Objet

Redessine entièrement le système de menus actif y compris tous les sous-menus actifs.

Commentaires

Cette routine efface tout d'abord le système de menus qui est affiché au moment de l'appel. Puis, elle redessine le système de menus, libérant tous les sous-menus qui étaient actifs initialement. Appelez cette procédure lorsque vous voulez simplement activer ou désactiver certains éléments de menu et que vous souhaitez faire apparaître les modifications.

Si le système de menus n'est pas visible, l'appel de *RetraceMenu* est inopérant.

Exemple

```
const
   TcheEcritureFich = 2;
var
   Mnu : Menu;
   ToucheM : ToucheMenu;
   Ch : Char;
   SortiePrete : Boolean;
begin
   {Définition du Menu}
   Mnu := NouvMenu([], Nil);
   ...
   ElemMenu('Ecriture de fichier', 2, 1, TcheEcritureFich, '');
   ...
   {Lecture d'un choix. Le système de menus reste à l'écran}
   ToucheM := ChoixMenu(Mnu, Ch);
   ...
   if not SortiePrete then begin
      {Désactive la sélection de Menu}
      DesactiveElemMenu(Mnu, TcheEcritureFich);
      {Reflète les modifications à l'écran}
      RetraceMenu(Mnu);
   end;
end.
```

Cet exemple met à jour l'écran après avoir désactivé un élément de menu.

Voir aussi

DesactiveElemMenu (TPMENU) ActiveElemMenu (TPMENU)

SelectElemMenu

Déclaration

```
procedure SelectElemMenu(Mnu : Menu; TcheM : ToucheMenu);
```

Objet

Recherche la touche TcheM dans le système de menus Mnu et redessine les menus pour la sélectionner.

Commentaires

Chaque élément d'un système de menus TPMENU doit comporter une valeur de sélection unique de type *ToucheMenu*. En utilisant cette routine, vous pouvez initialiser un système de menus de telle façon qu'une touche unique donnée devienne la touche de sélection courante du menu, quelle que soit sa position dans le système de menus.

SelectElemMenu vérifie d'abord que la touche spécifiée fait partie du système de menus et qu'elle est active. Si ce n'est pas la cas, *SelectElemMenu* s'arrête et ne fait rien. Si le système de menus est actuellement sur l'écran, et que l'élément de menu souhaité n'est pas dans le sous-menu courant, *SelectElemMenu* efface tout le système de menus. Ensuite, *SelectElemMenu* sélectionne la touche de menu spécifiée comme élément courant et redessine le système de menus pour l'afficher.

Exemple

```
SelectElemMenu(M, 10);
```

Sélectionne l'élément 10 du système de menus M et redessine le système de menus pour afficher l'élément sélectionné.

SousMenu

Déclaration

```
procedure SousMenu(XBP, YBP, YAideP : Byte; OrientP : Orientation;
                   CadreP : TabCadre; CouleursP : TabCouleurMenu;
                   ChnEntete : string);
```

Objet

Ajoute un sous-menu à un système de menus.

Commentaires

Lorsque vous définissez un nouveau système de menus, appelez tout d'abord *NouvMenu* et ensuite *SousMenu* une ou plusieurs fois pour définir les éléments de chaque élément d'un système de menus. Appelez *ElemMenu* pour ajouter les options désirées à chaque sous-menu.

XBP et *YBP* initialisent la position du coin supérieur gauche du sous-menu. Les valeurs autorisées pour *XBP* appartiennent à l'intervalle 1 / 80 et pour *YBP* à l'intervalle 1 / 25 (ou 43 ou 50). Si le sous-menu comporte un cadre, le coin supérieur gauche du cadre sera dessiné en *XBP, YBP*. S'il n'a pas de cadre, le texte du menu commencera à la même position.

Les sous-menus de TPMENU sont calculés automatiquement pour contenir les options, ainsi vous n'avez pas besoin de spécifier la position du coin inférieur droit d'un menu. Cependant, vous pouvez initialiser explicitement la largeur et la hauteur d'un sous-menu en appelant les routines *LargeurMenu* et *HauteurMenu* après l'appel de *SousMenu*.

YAideP initialise la position d'une ligne d'aide facultative pour le sous-menu. Chaque option du menu peut spécifier une ligne d'aide, affichée à partir de la colonne 1 de la ligne spécifiée. Pour désactiver cette caractéristique, spécifiez la constante *SansAide*.

OrientP spécifie l'orientation du sous-menu : horizontale ou verticale.

CadreP définit quel est le type des lignes encadrant le sous-menu. Voir la partie déclaration de *TabCadre* pour plus de détails sur le tableau de six caractères. Si vous voulez utiliser un menu de style Lotus sans cadre, passez la constante spéciale *CadreLotus* dans ce paramètre.

CouleursP définit les couleurs utilisées pour dessiner le sous-menu. TPMENU utilise six couleurs pour dessiner chaque sous-menu. Voir la partie déclaration de *TypeCouleurMenu* pour plus de détails. Si *FixeCouleurs* vaut "True", *CouleursP* correspondra aux meilleurs affichages monochrome et noir et blanc.

ChnEntete spécifie une chaîne de caractères affichée au centre de la barre supérieure d'un cadre de menu. Si cette chaîne est vide, aucun titre n'apparaîtra. Pour les menus de style Lotus, le titre est interdit.

SousMenu renvoie une erreur s'il n'y a pas assez de mémoire pour ajouter la structure du menu.

Exemple

```
SousMenu(1, 2, SansAide, Horizontal, Cadre1, Couleur1, 'TPMENU');
```

Définit un sous-menu commençant en colonne 1 ligne 2. Aucune aide ne sera affichée, le menu se déroulera horizontalement. Le cadre de menu et les couleurs sont enregistrés dans les constantes typées des types appropriés appelés *Cadre1* et *Couleur1*. Le titre TPMENU sera centré dans la barre supérieure du sous-menu.

TPEDIT : Editeur Ligne

La routine principale de l'unité TPEDIT est *LitChaine*, un éditeur de ligne sophistiqué compatible avec *WordStar* et comportant de nombreuses options. *LitChaine* offre les commandes d'édition suivantes :

(ENTER)	Entrée d'une ligne
(ESC), (CTRL)-(BREAK)	Quitte sans modifier la ligne
(←), (CTRL)-S	Déplace le curseur d'un caractère vers la gauche
(→), (CTRL)-D	Déplace le curseur d'un caractère vers la droite
(CTRL)-(←), (CTRL)-A	Déplace le curseur d'un mot vers la gauche
(CTRL)-(→), (CTRL)-F	Déplace le curseur d'un mot vers la droite
(HOME), (CTRL)-Q-S	Déplace le curseur au début de la ligne
(END), (CTRL)-Q-D	Déplace le curseur à la fin de la ligne
(DEL), (CTRL)-G	Efface le caractère sous le curseur
(BACKSPACE), (CTRL)-H, (CTRL)-(BACKSPACE)	Efface le caractère à gauche du curseur
(CTRL)-(END), (CTRL)-Q-Y	Efface la fin de la ligne
(CTRL)-Y, (CTRL)-X	Efface toute la ligne
(CTRL)-(HOME)	Efface à partir du début de la ligne
(CTRL)-T	Efface le mot à droite du curseur
(INS)	Mode insertion on ou off. Un curseur plein indique un mode insertion, un petit curseur indique le mode recouvrement.
(CTRL)-R, (CTRL)-Q-L	Restaure le contenu initial d'une ligne.
(CTRL)-P	Si vous activez cette commande, le prochain caractère entré sera inséré dans la chaîne. Si c'est un caractère de contrôle, et que *EditeCtrls* vaut "True", il sera affiché en caractère alphabétique majuscule avec les couleurs spécifiées par le paramètre *AttrControle* de *LitChaine* plutôt que par le paramètre *AttrChn*. Par exemple, (CTRL)-P-(CTRL)-C affichera un "C" avec *AttrControle*. Si *EditeCtrls* vaut "False", le caractère de contrôle sera affiché normalement (avec *AttrControle*). Lors de l'attente d'un caractère de contrôle,

LitChaine affichera temporairement un curseur en forme de bloc. Une fois qu'une touche est pressée, le curseur reprend sa forme précédente. Deux points méritent d'être notés : premièrement, si *AttrControle* = *AttrChn* et *EditeCtrls* = *True*, cette commande sera ignorée ; deuxièmement, aucune des routines de lecture, sauf *LitChaine*, ne comporte de paramètre *AttrControle* puisque les caractères de contrôle ne devraient jamais être considérés comme des données d'entrée valides lors de la saisie de nombres.

TPEDIT fournit également un support de la souris qui peut être activé simplement en appelant *ActiveEditSouris*. Si le support de la souris est activé, les commandes suivantes sont disponibles avec *LitChaine* ou n'importe quel ordre de lecture numérique.

Cliquer à gauche	Equivalent à presser (ENTER)
Cliquer à droite	Equivalent à presser (ESC)
Cliquer à gauche et à droite	Equivalent à presser (F1)

LitCaractere et *OuiOuNon* peuvent également être activées lors de l'utilisation du bouton de la souris. Par défaut, le bouton gauche est équivalent à la touche (ENTER) et tout autre bouton ou combinaison de boutons équivaut à la frappe de la touche (ESC). Mais, vous pouvez changer les relations touches/boutons en modifiant la table de conversion *EditeCodesBouton*.

Vous pouvez modifier les touches d'édition et les affectations des fonctions de la souris avec l'une des trois méthodes décrites en appendice C. De plus, il y a de nombreuses options de modification que vous pouvez sélectionner en modifiant simplement une constante globale typée : forcer des majuscules (*ForceMajuscule* par défaut égale à "False") ; démarrer la modification avec le curseur au début ou à la fin d'une chaîne (*CursEnFin* par défaut en fin de chaîne) ; éliminer automatiquement des blancs de tête ou de queue (*EffEspaces* par défaut égal à "True") ; effacer la chaîne si le premier caractère entré est un caractère ASCII (*EffacePremCarac*, valeur par défaut égale "True") ; considèrer les coordonnées écran comme des coordonnées relatives dans la fenêtre actuelle (*FenRelative*, valeur par défaut égale "False") ; les caractères de contrôle correspondent à des caractères alphabétiques majuscules (*EditeCtrls*, valeur par défaut égale "True") ; les caractères entrés ne sont pas affichés (*ModeMotDePasse*, valeur par défaut égale "False") ; le mode d'édition par défaut pour l'insertion et le recouvrement (*InsParDefaut*, valeur par défaut égale "True").

Puisque les routines de modification de chaînes numériques -- *LitLongEntier*, *LitEntier*, *LitMot*, et *LitReel* -- utilisent ensemble *LitChaine*, les commandes de modification disponibles dans ces dernières sont identiques. La seule chose qui les distingue dans une optique utilisateur est qu'elles effectuent des contrôles. Si l'utilisateur entre une chaîne incorrecte ou une valeur en dehors des limites que vous spécifiez, un message d'erreur est affiché et l'utilisateur peut à nouveau entrer la donnée.

Les deux autres routines de TPEDIT, *LitCaractere* et *OuiOuNon* sont utilisées pour afficher des caractères individuels. *LitCaractere* vous permet de limiter l'ensemble des caractères valides que vous pouvez entrer. *OuiOuNon* limite normalement l'ensemble des caractères valides à 'O' et 'N'. Mais si vous affectez une de ces valeurs au paramètre *Défaut*, elles afficheront des réponses par défaut dans un encadré tel que :

```
Voulez-vous continuer ? [O]
```

et permettront à l'utilisateur de sélectionner des valeurs par défaut en appuyant sur (ENTER).

Remarques pour les utilisateurs de la version 4 de Turbo Professionnel

Si vous possédez une version antérieure à la version 4.04, une légère modification de *LitChaine* (et les routines qui l'appellent telles que *LitReel*) peut affecter les applications existantes. Pour expliquer cette modification, prenons un exemple. Supposons une longueur de chaîne maximum égale à 10 caractères (via le paramètre *LongMax* de *LitChaine*), *LitChaine* utilisera 11 colonnes de l'écran, vous pourrez entrer 10 caractères et il vous restera une position à gauche (dans laquelle se placera le curseur lorsque la chaîne aura la longueur maximum). Avec les versions antérieures à 4.04, *LitChaine* réservait seulement 10 colonnes et plaçait le curseur un caractère à droite de la zone d'édition. Cette modification améliore la présentation de l'affichage et évite certains problèmes en colonne 80 de l'écran. Vous pourrez simuler l'ancienne version en initialisant la constante globale typée *PlaceCursEnFin* à "False".

TPEDIT : Déclarations

Constantes

```
EffacePremCarac : Boolean = True;
```

Si cette constante vaut "True", *LitChaine* et les lectures numériques effaceront la chaîne existante si le premier caractère entré est un caractère ASCII.

```
CursEnFin : Boolean= True;
```

Si cette constante vaut "True", la fonction *LitChaine* et les fonctions de lecture numérique (*LitLongEntier, LitEntier, LitMot, LitReel*) placent le curseur à la fin de la chaîne en début de modification plutôt qu'au début de la chaîne.

```
EditeCtrls : Boolean = True;
```

Si cette constante vaut "True", tous les caractères de contrôle entrés dans une chaîne seront convertis en caractères alphabétiques majuscules et affichés avec l'attribut contenu dans le paramètre *AttrControle* de *LitChaine*. Si elle vaut "False", ils sont affichés normalement (avec *AttrControle*).

```
TailleEdit : Byte = 0;
```

Lorsque *TailleEdit* vaut zéro, le paramètre *LongMax* passé à *LitChaine* définit à la fois la largeur maximum de la chaîne et la largeur du champ d'édition. Si vous initialisez *TailleEdit* à une valeur différente de zéro avant d'appeler *LitChaine*, cette valeur sera utilisée pour définir la largeur du champ d'édition, alors que le paramètre *LongMax* spécifie la longueur maximum de la chaîne. *TailleEdit* vous permet donc de modifier des chaînes de caractères plus longues que la largeur de l'écran. Par exemple, si vous voulez modifier une chaîne de 128 caractères, mais seulement avec 40 colonnes écran pour le champ d'édition, vous pouvez appeler *LitChaine* comme suit :

```
TailleEdit : 40;
LitChaine(Mes, Ligne, Col, 128, Attr, Attr, Attr, Esc, C);
TailleEdit := 0;
```

LitChaine utilisera alors seulement 40 colonnes pour le champ d'édition et, opérera un déroulement horizontal éventuellement.

```
ForceMajuscule : Boolean = False;
```

Si cette constante vaut "True", *LitChaine* et les fonctions numériques de lecture forcent les caractères en majuscules.

```
CacheCursDansLitCarac : Boolean= False;
```

Si cette constante vaut "True", *LitCaractere* et *OuiOuNon* cachent temporairement le curseur.

```
PlaceCursEnFin : Boolean = True ;
```

Si ce paramètre vaut "True", *LitChaine* étend la zone d'édition d'une position laissant un espace pour y placer le curseur lorsque la chaîne à modifier est complète.

```
InsParDefaut : Boolean = True;
```

Si cette constante vaut "True", *LitChaine* et les fonctions de lecture numérique utilisent par défaut le mode insertion plutôt que le mode recouvrement.

```
LCcaracCompl : Char = ' ';
```

Ce caractère est utilisé pour compléter la chaîne à modifier lorsqu'elle est affichée. Un autre caractère possible est '_'.

```
AfficheCarLu : Boolean = False;
```

Si cette constante vaut "True", *LitCaractere* et *OuiOuNon* affichent le caractère sélectionné.

```
ModeMotDePasse : Boolean = False;
```

Si ce paramètre vaut "True", *LitChaine* n'affichera pas les caractères de la chaîne à modifier. Chaque caractère sera représenté sur l'écran comme *CaracMotDePasse* (voir ci-dessous).

```
CaracMotDePasse : Char = ^G;
```

Caractère utilisé pour représenter le texte en mode mot de passe. Le caractère ^G est un point.

```
LCrien       = 00;        {n'est pas une commande}
LCcarac      = 01;        {caractère pour entrer la chaîne}
LcCaracCtrl  = 02;        {accepte caractère de contrôle}
LCsaisie     = 03;        {accepte la chaîne actuelle et quitte}
LCquitte     = 04;        {restaure la chaîne par défaut et quitte}
LCrestaure   = 05;        {restaure la valeur par défaut et continue}
LChome       = 06;        {curseur en début de ligne}
LCend        = 07;        {curseur en fin de ligne}
LCgauche     = 08;        {curseur d'un caractère à gauche}
LCdroite     = 09;        {curseur d'un caractère à droite}
LCmotGauche  = 10;        {curseur d'un mot à gauche}
LCmotdroit   = 11         {curseur d'un mot à droite}
LCback       = 12;        {retour arrière d'un caractère}
LCdel        = 13;        {efface le caractère sous le curseur}
LCeffFinLg   = 14         {efface la fin de ligne}
LCeffDebLg   = 15;        {efface à partir du début de ligne}
LCeffLg      = 16;        {efface toute la ligne}
LCeffMot     = 17;        {efface le mot à droite du curseur}
```

```
LCins        = 18          {mode insertion}
LCaide       = 19;         {invoque une routine d'aide}
LCutl0       = 20;         {commande sortie utilisateur}
...
LCutl9       = 29;
```

Codes pour chacune des commandes d'édition de *LitChaine*. La plupart d'entre-eux s'expliquent d'eux-mêmes. Les dix derniers (LCutl0..LCutl9) permettent de définir des commandes de sortie utilisateur. Si l'une de ces commandes est exécutée, *LitChaine* se comportera comme si vous aviez pressé (ENTER). Vous aurez alors à contrôler la variable globale *LCcommande* et à exécuter la fonction associée avec la commande sélectionnée. Pour plus de détails sur la manière d'implémenter des commandes utilisateur, référez-vous à l'appendice C.

```
EffEspaces : Boolean= True;
```

Si cette constante vaut "True", *LitChaine* élimine automatiquement les blancs au début et à la fin (temporairement pour les routines de lecture de valeurs numériques).

```
FenRelative : Boolean = False;
```

Si *FenRelative* vaut "True", toutes les routines de TPEDIT interpréteront les coordonnées écran qui leur seront passées comme des coordonnées relatives dans la fenêtre courante plutôt que comme des coordonnées absolues. C'est au programmeur de s'assurer que les champs d'édition se cadrent bien dans la fenêtre actuelle.

Types

```
EnsembleCarac = set of char;
```

Caractères de base

```
{$IFDEF UseMouse}
EditeCodesBouton : array[$E9..$EF] of word = (
   $011B,    {Les trois boutons          = <Esc>}
   $011B,    {boutons droit et central   = <Esc>}
   $011B,    {boutons gauche et central  = <Esc>}
   $011B,    {bouton central             = <Esc>}
   $011B,    {Deux boutons               = <Esc>}
   $011B,    {bouton droit               = <Esc>}
   $1C0D,    {bouton gauche              = <Entrée>}
{$ENDIF}
```

Cette table est utilisée par *LitCaractere* et *OuiOuNon* afin de convertir les boutons de la souris en codes touches. Comme vous pouvez le voir, la frappe du bouton gauche de la souris donne le même résultat que la frappe de (ENTER), alors que la frappe des autres boutons ou une combinaison de ces boutons produit le même résultat que la frappe de (ESC). Vous êtes naturellement libre de modifier cette table si vous le souhaitez.

```
LCType = LCrein..LCUtl9;
```

Commande de modification de *LitChaine*.

Variables

```
PtrToucheEdit : Pointer;
```

Pointe sur une routine qui renverra la prochaine touche. Voir l'appendice C pour plus de détails.

```
PtrAideEdit : Pointer;
```

Pointe sur une routine à appeler lorsque la commande d'aide est donnée. Voir l'appendice C pour plus de détails.

```
EditeIndexAide : Word;
```

Thème d'aide courant. Cette valeur est passée à la routine pointée par *PtrAideEdit*. Voir l'appendice C pour plus de détails.

```
LCCommande : LCType;
```

La dernière commande d'édition donnée dans *LitChaine* (ou n'importe quelle entrée numérique).

```
LCMotCar : Word;
```

Dernière touche entrée dans *LitChaine* (ou n'importe quelle lecture numérique). En contrôlant la valeur de cette variable, vous pouvez déterminer la touche qui a été utilisée pour sortir de l'éditeur : (ENTER), (ESC), (CTRL)-(BREAK).

ActiveEditSouris

Déclaration

```
procedure ActiveEditSouris;
```

Objet

Active le support de la souris dans TPEDIT.

Commentaires

Cette routine active le support de la souris dans TPEDIT. Outre l'initialisation de l'indicateur interne pour utiliser les autres routines de TPEDIT, elle active le gestionnaire d'événements de la souris de TPSOURIS et initialise *PtrToucheEdit* pour qu'il pointe sur *LitToucheOuBouton* dans TPSOURIS. Voir l'introduction de TPSOURIS pour plus de détails.

Voir aussi

ActiveAttenteEvenement (TPSOURIS)
LitToucheOuBouton (TPSOURIS)

DesactiveEditSouris

Déclaration

```
procedure DesactiveEditSouris;
```

Objet

Désactive le support de la souris dans TPEDIT.

Commentaires

Cette routine désactive le support de la souris dans TPEDIT. Outre l'effacement de l'indicateur interne initialisé par *ActiveEditSouris*, elle désactive le gestionnaire d'événements de la souris de TPSOURIS et initialise *PtrToucheEdit* pour pointer sur *LitMotTouche* (TPCRT)

LitCaractere

Déclaration

```
procedure LitCaractere(Message : string; Lig, Colonne : Byte;
                       AttrMessage : Byte; Accepte : EnsembleCarac ;
                       var Car : Char);
```

Objet

Affiche un message et attend la frappe d'une touche.

Commentaires

Cette fonction fournit un moyen pratique d'interroger l'utilisateur et de lui permettre d'entrer un caractère qui sera vérifié par rapport à un intervalle de caractères acceptables.

Paramètres : *Message* est une chaîne de caractères affichée sur l'écran en position *Lig, Colonne. AttrMessage* est l'attribut vidéo utilisé pour afficher *Message. Accepte* est l'ensemble des caractères acceptables.

Le premier caractère acceptable saisi est renvoyé dans *Car*. Tous les caractères sont automatiquement convertis en majuscules, ainsi "['A'..'Z']" est traité comme étant équivalent à "['a'..'z', 'A'..'Z']". Les codes de balayage étendus sont ignorés.

Si vous devez utiliser *LitCaractere* pour lire des touches de fonction, des touches du curseur etc..., consultez les explications de la routine *ChoixMenu* dans TPMENU. *LitCaractere* utilise la même technique de traitement des bits que TPMENU pour convertir les touches étendues sur un seul octet.

Notez que les constantes globales typées *CacheCursDansLitCarac* et *AfficheCarLu* vous permettent de personnaliser *LitCaractere*.

Exemple

```
var Ch : Char;
begin
   AfficheCarLu := True;
   LitCaractere('Entrez une lettre de l''alphabet : ', 1, 1, $07,
         ['A'..'Z'], Ch);
   LitCaractere('Entrez un chiffre (0..9) : ', 2, 1, $07, ['0'..'9'],
                  Ch);
   LitCaractere('Entrez n''importe quel caractère : ', 3, 1, $07,
         [#0..#255], Ch);
end.
```

Utilisation de *LitCaractere* pour entrer un caractère alphabétique, numérique ou n'importe quel caractère.

Voir aussi

OuiOuNon (TPEDIT) ChoixMenu(TPMENU)

LitChaine

Déclaration

```
procedure LitChaine(Message : string; Lig, Colonne, LongMax : Byte;
                    AttrMessage, AttrChn, AttrControle : Byte;
                    var Esc : Boolean; var S : string);
```

Objet

Entrée d'une chaîne de caractères.

Commentaires

La routine principale de TPEDIT, *LitChaine,* est celle d'un éditeur de ligne offrant une compatibilité totale avec *WordStar* ainsi que plusieurs options utiles qui vous permettent de contrôler sa mise en oeuvre.

Message est une chaîne de caractères affichée sur l'écran en *Lig, Colonne* (*Lig* et *Colonne* sont considérées comme coordonnées absolues à moins que la constante typée globale *FenRelative* n'ait la valeur "True"). *S* est la chaîne à modifier qui est affichée à sa droite. *LongMax* est la longueur maximum de la chaîne (dans l'intervalle 1..254). *AttrMessage* est l'attribut vidéo utilisé pour afficher *Message. AttrChn* est l'attribut vidéo utilisé pour afficher la chaîne de caractères à modifier et *AttrControle* est l'attribut utilisé pour afficher les caractères de contrôle. Si vous pressez (ESC), *S* est inchangée et *Esc* prend la valeur "True".

Remarquez que les constantes globales typées *ForceMajuscule, CursEnFin, EffEspaces, EditeCtrls, ModeMotDePasse* et *InsParDefaut* vous permettent de personnaliser le comportement de *LitChaine.*

Pour une liste complète des commandes de modifications reconnues par *LitChaine,* voir les généralités au début de cette unité.

Exemple

```
var Nom,Prenom,Adresse : String; Escape : boolean;
begin
   Nom := 'Mélusine Enfaillite';
   LitChaine('Nom: ',1,1,30,$07,$70,$0F,Escape,Nom);
   if Escape  then Exit;
   Surnom := 'Sans Travail';
   LitChaine('Prenom: ',2,1,30,$07,$70,$0F,Escape,Prenom);
   if Escape  then Exit;
   Adresse := '1 rue de la Rêverie';
   LitChaine('Adresse: ',3,1,25,$07,$70,$0F,Escape,Adresse);
   if Escape  then exit;
end;
```

Voir aussi

EditeChaine(TPSAISIE)

LitEntier

Déclaration

```
procedure LitEntier(Message : string; Lig, Colonne, LongMax : Byte;
                    AttrMessage, AttrChn : Byte; NInf, NSup : Integer;
                    var Escape : Boolean; var N : Integer);
```

Objet

Demande d'entrée d'un entier, compris dans l'intervalle *NInf..NSup*.

Commentaires

LitEntier est semblable à *LitLongEntier*, mais le nombre entré doit appartenir à l'intervalle -32768..32767 si *NInf* (borne minimum acceptable) et *NSup* (borne supérieure acceptable) sont à zéro.

Exemples

Voir les explications de *LitLongEntier* (TPEDIT) pour des exemples.

Voir aussi

LitLongEntier (TPEDIT)
LitChaine (TPEDIT)
LitMot (TPEDIT)

LitLongEntier

Déclaration

```
procedure LitLongEntier(Message : string;
                        Lig, Colonne, LongMax : Byte;
                        AttrMessage, AttrChn : Byte;
                        LInf, LSup : LongInt;
                        var Esc : Boolean; var L : LongInt);
```

Objet

Demande de saisie d'un entier long dans l'intervalle *LInf..LSup*.

Commentaires

Cette fonction offre un moyen pratique de saisir des entrées sous forme d'une chaîne de caractères représentant un entier long. *LitLongEntier* convertit la valeur par défaut (*L*) en une chaîne de caractères et permet à l'utilisateur de modifier la chaîne puis convertit en retour la chaîne de caractères en un entier long renvoyé dans *L*. Si l'utilisateur presse (ESC), *L* est inchangé et *Esc* prend la valeur "True".

Les autres paramètres : *Message* est une chaîne de caractères affichée sur l'écran en *Lig, Colonne* (la chaîne à modifier est affichée à sa droite), *LongMax* est la longueur maximum de la chaîne à entrer (par exemple, la largeur du champ de modifications), *AttrMessage* est l'attribut vidéo utilisé pour afficher le message et *AttrChn* est l'attribut utilisé pour afficher la chaîne de caractères à modifier.

Si *LInf* (limite inférieure acceptable) et *LSup* (limite supérieure acceptable) sont égales à zéro, le nombre entré est dans l'intervalle -2147483648..2147483647. Sinon, le nombre se trouve dans l'intervalle *LInf..LSup*. Si un nombre incorrect est saisi, un message d'erreur est affiché (si *LongMax* est plus grand ou égal à la longueur du message d'erreur), ou un bip est émis.

Dans les autres cas, l'utilisateur peut corriger le nombre à saisir. Remarquez que les constantes globales typées *ForceMajuscule, CursEnFin* et *InsParDefaut* vous permettent de personnaliser le comportement de *LitLongEntier*.

Exemple

```
var
   L : LongInt;
   Esc : boolean;
begin
   L := 0;
   LitLongEntier('Entrez un entier long : ', 1, 1, 11, $07, $0F, 0, 0,
                 Esc, L);
   if Esc then Exit;
   L := 0;
   LitLongEntier('Entrez un nombre compris entre -100 et 100 : ', 2,
                 1, 4, $07, $70, -100, 100, Esc, L);
   if Esc then Exit;
end.
```

Voir aussi

LitChaine (TPEDIT).

LitMot

Déclaration

```
procedure LitMot(Message : string; Lig, Colonne, LongMax : Byte;
                 AttrMessage, AttrChn : Byte; MInf, MSup : Word;
                 var Esc: Boolean; var W : Word);
```

Objet

Demande de saisie d'un entier dans l'intervalle *MInf..MSup*.

Commentaires

LitMot est semblable à *LitLongEntier* mais le nombre entré est restreint à l'intervalle 0..65535 si *MInf* (limite inférieure acceptable) et *MSup* (limite supérieure acceptable) sont égales à zéro.

Exemples

Voir les explications de *LitLongEntier* (TPEDIT) pour des exemples.

Voir aussi

LitEntier (TPEDIT)
LitLongEntier (TPEDIT)
LitChaine (TPEDIT)

LitReel

Déclaration

```
procedure LitReel(Message : string;
                  Lig, Colonne, LongMax, AttrMessage,
                  AttrChn, PositionsDec : Byte; RInf, RSup : Real;
                  var Esc : Boolean; var R : Real);
```

Objet

Demande de saisie d'un réel dans l'intervalle *RInf..RSup*.

Commentaires

Cette fonction offre un moyen pratique de saisie d'une chaîne de caractères représentant un nombre réel. *LitReel* convertit la valeur par défaut (*R*) en une chaîne de caractères et permet à l'utilisateur de modifier la chaîne puis convertit la chaîne en un réel renvoyé dans *R*. Si l'utilisateur presse (ESC), *R* est inchangé et *Esc* prend la valeur "True".

Autres paramètres : *PositionsDec* définit le formatage de *R* lorsqu'il est converti initialement en chaîne de caractères. Par exemple, si *R* vaut 1.2345 et que *PositionsDec* vaut 2, *R* sera affichée comme '1.23'. *Message* est une chaîne affichée sur l'écran en *Lig, Colonne* (la chaîne à modifier est affichée à sa droite). *LongMax* est la longueur maximum de la chaîne à saisir (par exemple, la largeur du champ). *AttrMessage* est l'attribut vidéo utilisé pour afficher *Message* et *AttrChn* est l'attribut vidéo utilisé pour afficher la chaîne de caractères à modifier.

R, *RInf* et *RSup* sont de type Float, défini dans l'unité TPCHAINE. Si TPCHAINE est compilée avec l'option $N-, il s'agit d'un réel flottant sur 6 octets. Si TPCHAINE est compilée avec l'option $N+, il s'agit d'un réel extended (Real).

Si *RInf* (limite inférieure acceptable) et *RSup* (limite supérieure acceptable) sont égales, aucun contrôle de bornes n'est effectué et n'importe quel nombre réel valide peut être saisi. Sinon, le nombre est restreint à l'intervalle *RInf.. RSup*. Si un nombre incorrect est saisi, un message d'erreur est affiché (si *LongMax* est supérieur ou égal à la longueur du message) ou un bip est émis. Dans les deux cas, l'utilisateur peut corriger le nombre à saisir.

Remarquez que les constantes globales typées *ForceMajuscule, CursEnFin* et *InsParDefaut* vous permettent de personnaliser le comportement de *LitReel*.

Exemple

```
var R : Real;
   Esc : Boolean;
begin
   R := 0.0;
   LitReel('Entrez une nombre quelconque : ',
            1,1,20,$07,$0F,2,0.0,0.0,Esc,R);
   if Esc then Exit;
   LitReel('Entrez une nombre compris entre 0.0 et
            100.0 : ',2,1,5,$07,$70,
            1,0.0,100.0,Esc,R);
   if Esc then Exit;
end.
```

Voir aussi

LitChaine (TPEDIT) Float(TPCHAINE)

OuiOuNon

Déclaration

```
function OuiOuNon(Message : string; Lig, Colonne : Byte;
                  AttrMessage : Byte; Defaut : Char) : Boolean;
```

Objet

Renvoie "True" pour oui, "False" pour non.

Commentaires

Cette fonction founit le moyen de demander à l'utilisateur de saisir une réponse Oui ou Non.

Paramètres : *Message* est une chaîne de caractères affichée sur l'écran en *Lig, Colonne*. *AttrMessage* est l'attribut vidéo utilisé pour afficher *Message*. La valeur du paramètre *Defaut* peut être n'importe quel caractère ; normalement *OuiOuNon* acceptera seulement **O** ou **N** comme réponse. Quoiqu'il en soit, si *UpCase* (*Defaut*) est **O** ou **N**, la fonction *OuiOuNon* le traitera comme le choix par défaut et ajoutera '[O]' ou '[N]' à la fin du message et supposera que (ENTER) signifie que l'utilisateur accepte la valeur par défaut. La fonction renvoie "True" si **O** est entré (ou la valeur par défaut O est acceptée) et "False" si **N** est entré (ou si la valeur par défaut N est acceptée).

Remarquez que les constantes globales typées *CacheCursDansLitCarac* et *AfficheCarLu* vous permettent de personnaliser le comportement de *OuiOuNon*.

Exemples

```
if not OuiOuNon('Voulez-vous continuer?',1,1,$07,#0) then Exit;
```

Affiche 'Voulez-vous continuer ?' et attend la frappe de **O** ou **N**.

```
if not OuiOuNon('Voulez-vous continuer?',1,1,$07,'O') then Exit;
```

Affiche 'Voulez-vous continuer ? [O]' et attend la frappe de **O**, **N** ou (ENTER) . Si l'utilisateur presse (ENTER) la fonction retournera "True".

```
if not OuiOuNon('Voulez-vous continuer?',1,1,$07,'N') then Exit;
```

Affiche 'Voulez-vous continuer ? [N]' et attend la frappe de **O**, **N** ou (ENTER) . Si l'utilisateur presse (ENTER) la fonction renverra "False".

Voir aussi

LitCaractere (TPEDIT)

TPAIDE : Aide adaptée au contexte

Si vous avez déjà développé un système d'aide adaptée au contexte dans une application, vous savez qu'il faut beaucoup d'efforts pour construire un système efficace et facile à utiliser. L'unité TPAIDE est un outil simple et puissant permettant d'ajouter des écrans d'aide dans vos programmes. Votre travail consistera essentiellement à écrire les textes d'aide.

Le produit se compose réellement de deux parties : une unité TPAIDE.TPU et un "compilateur de fichiers d'aide" CREEAIDE.EXE qui convertit le fichier texte d'aide en fichier binaire indexé utilisé par TPAIDE.

Le fichier binaire peut être lié directement dans le programme hôte. Néanmoins pour les gros fichiers d'aide ou pour les applications importantes, le fichier d'aide peut être externe et lu en mémoire à la demande par TPAIDE.

Le système d'aide supporte au maximum 10 000 thèmes clés en un seul fichier, chaque thème pouvant contenir au plus vingt pages d'informations. Chaque thème clé peut contenir jusqu'à 50 références croisées vers d'autres thèmes. Les textes d'aide peuvent être affichés avec 6 attributs vidéo différents. Le compilateur coupe et pagine le texte d'aide initial à partir des directives que vous lui donnez.

L'aide peut être sélectionnée de différentes façons à partir du programme : par index, par nom, dans une liste Pick, dans un index maître, par références croisées ou par référence dans une pile d'éléments déjà sélectionnés.

La plupart des ces options peuvent être mises en oeuvre automatiquement. Grâce à la souris, TPAIDE permet de créer des systèmes d'aide plus conviviaux.

Avant d'expliquer séparément les procédures et les type de l'unité TPAIDE, décrivons la méthode de développement d'un système d'aide.

Un exemple simple

Cet exemple illustre une méthode d'utilisation d'un système d'aide dans un programme.

```
program DemoAide;
uses
   Dos, TpDos, TpChaine, TpCrt, TpFenetr, TpCmd, TpPick, TpAide;
const
   CouleursAide : TabCouleurAide =
                          ($07, $07, $0F, $70, $09, $70, $0F, $01);
var
   Aide : PtrAide;
   Statut : Word;
   Theme : Word;
begin
   ClrScr;
   Statut := OuvreFicAide('DEMO.HLP',      {Nom du fichier d'aide
                                            binaire}
                          10, 2,           {Coin supérieur gauche
                                            de la fenêtre d'aide}
                          22, 2,           {Ligne inférieure et
                                            colonnes de l'index d'aide}
                          CouleursAide,    {Couleurs à utiliser}
                          Aide);           {Descripteur d'aide}
   if Statut <> 0 then begin
      case Statut of
           2 : WriteLn('Fichier d''aide DEMO.HLP introuvable');
         100 : WriteLn('Fin de fichier inattendue dans DEMO.HLP');
         106 : WriteLn('Le fichier d''aide DEMO.HLP n''est pas
                        valide');
         203 : WriteLn('Espace dans le tas insuffisant pour le système
                       d''aide');
      else
         WriteLn('Erreur d''initialisation d''aide ', Statut);
      end;
      Halt;
   end;

   {Personnalisation de la fenêtre d'aide}
   TpFenetr.Ombre := True;
   TpFenetr.AttrOmbre := $70;

   repeat
      {Choix d'un thème dans l'index d'aide}
      Theme := ListeAide(Aide, 2, 2, 10, 1);
         if Theme <> 0 then
            {Affiche l'aide concernant ce thème}
            if not AffAide(Aide, Theme) then
               WriteLn('Impossible d''afficher l''aide concernant le
                       thème', Theme);
   until Theme = 0;

   {Ferme le fichier d'aide et libère la mémoire}
   FermeAide(Aide);
end.
```

CouleursAide est une table d'attributs vidéo utilisée pendant l'affichage de l'information d'aide. Nous examinerons le rôle de chaque élément plus tard. Vous aurez probablement besoin de deux jeux d'attributs : un pour un moniteur couleur, l'autre pour un moniteur monochrome.

La variable *Aide* sert d'interface entre votre programme et TPAIDE. Cette variable pointe sur un enregistrement contenant toutes les données sur votre système d'aide nécessaires à TPAIDE. La variable *Aide* peut être initialisée par un appel à *OuvreFicAide* ou *OuvreMemAide*.

Notre exemple lit le fichier d'aide DEMO.HLP généré par l'utilitaire CREEAIDE.

Notez les coordonnées qui sont passées à *OuvreFicAide*. Les deux premières valeurs numériques correspondent au coin supérieur gauche de la fenêtre d'aide, y compris éventuellement son cadre.

La valeur suivante correspond à la ligne inférieure de la fenêtre d'aide, y compris le cadre. La largeur de la fenêtre d'aide est inscrite dans le fichier d'aide lorsque CREEAIDE découpe le texte et il n'est pas nécessaire de passer le paramètre relatif à la colonne droite à la procédure *OuvreFicAide*. Le dernier paramètre correspond au nombre de colonnes des éléments utilisés par l'index d'aide. Si l'utilisateur presse F1 alors que l'information d'aide est à l'écran, TPAIDE redessine automatiquement la fenêtre d'aide pour afficher la liste de tous les thèmes d'aide indexés.

En fonction de la largeur du texte d'aide et du nombre de texte, vous pouvez afficher une, deux ou plusieurs colonnes côte-à-côte.

OuvreFicAide recherche d'abord DEMO.HLP (dans le répertoire courant et dans le path du DOS). *OuvreFicAide* ouvre ensuite le fichier et vérifie s'il est dans le bon format. Finalement, la procédure alloue un buffer sur le tas et renvoie un statut de type Word. Dans notre exemple, nous contrôlons ce statut et arrêtons s'il contient un code d'erreur.

Nous pouvons tirer profit de toutes les techniques de formatage disponibles, puisque TPAIDE utilise les routines de TPFENETR. Dans cet exemple, nous utilisons des fenêtres ombrées.

Ensuite, nous appelons la routine *ListeAide* de TPAIDE qui nous permet de sélectionner un élément dans la liste des thèmes affichés. *ListeAide* renvoie le numéro du thème ou la valeur 0 si aucun thème n'est sélectionné.

AffAide affiche alors une fenêtre contenant le texte d'aide sur ce thème. L'utilisateur peut parcourir les pages d'information ou sélectionner un autre thème. L'écran est restauré à son état précédent à la sortie de *AffAide*.

Enfin, le programme appelle *FermeAide* qui ferme le fichier d'aide et libère toute la mémoire allouée pour le système d'aide.

Directives de fichiers d'aide

L'information affichée par le système d'aide est celle que vous avez d'abord écrit dans un fichier texte. Vous pouvez y ajouter des directives pour indiquer au compilateur comment formater et interpréter votre fichier texte. Vous pouvez ensuite le compiler avec CREEAIDE pour créer un fichier formaté indexé et compacté qui peut être lu par TPAIDE.

Nous utilisons l'extension .TXT pour un fichier texte d'aide et réservons le suffixe .HLP pour le fichier binaire généré par CREEAIDE. Le système d'aide ne l'exige cependant pas.

Les directives commencent par un point d'exclamation, immédiatement suivi par le nom de la directive, puis par un ou plusieurs paramètres optionnels séparés par un ou plusieurs blancs. Voici la liste des directives reconnues par le compilateur :

!LARGEUR largeur

Spécifie le nombre de colonnes de la fenêtre d'aide y compris éventuellement le cadre. Le texte pourra être affiché avec une colonne vide pour le séparer des bords du cadre. Dans ce cas, la largeur réelle du texte est donc largeur - 4 colonnes. Si cette directive est absente, la valeur par défaut de largeur est 40.

Comme CREEAIDE découpe le texte, la largeur est définie dans le fichier d'aide lui-même. Elle ne peut plus être modifiée lors de l'affichage. Au contraire, CREEAIDE n'insère pas automatiquement les sauts de page dans le texte d'aide, ce qui permet d'adapter automatiquement les écrans au mode texte 25, 43 et 50 lignes.

La directive !LARGEUR doit précéder la première directive !THEME.

Exemple

```
!LARGEUR 70
```

CREEAIDE découpe le texte en ligne de 66 caractères maximum.

!THEME Numéro [Nom]

Chaque thème d'aide doit commencer par une directive !THEME. Les thèmes d'aide seront entrés dans l'ordre numérique croissant en commençant au numéro 1. Cette directive peut contenir un nom de thème facultatif. Le nom apparaîtra tel qu'il a été saisi avec la directive et peut contenir des espaces (ce nom sera affiché comme titre de fenêtre d'aide et optionnellement comme point d'entrée de l'index des thèmes).

L'espace utilisé pour les *Noms* dans le fichier d'aide est limité à 64 Ko ; l'espace réservé est égal à la longueur du plus grand nom plus le nombre total de thèmes d'aide. Si les noms de thèmes sont utilisés, l'espace est

limité par le nombre maximum de thèmes dans le fichier d'aide. CREEAIDE vérifie que la limite de 64 K n'est pas dépassée.

Les lignes (même vides) qui suivent la directive !THEME, marquent le commencement du texte pour le texte considéré. Le texte d'aide se termine à la rencontre de la directive !THEME suivante ou lorsque la fin du fichier .TXT est atteinte.

Chaque fichier d'aide doit contenir au moins une directive !THEME. Le thème de numéro 0 est interdit, il est réservé par le système d'aide pour mémoriser les erreurs.

Exemple

```
!THEME 1 Première partie
!THEME 22
```

Le thème 1 comporte un nom qui sera affiché comme titre de la fenêtre d'aide. La fenêtre d'aide du thème 22 ne comportera pas de titre et ce thème n'apparaîtra dans aucun index.

!INDEX NumeroIndex
!SANSINDEX

Ces directives facultatives permettent d'associer des thèmes particuliers à l'index des thèmes et doivent apparaître après chaque directive !THEME.

Si ces directives sont absentes, CREEAIDE indexe chaque nom de thème et trie l'index alphabétiquement.

Un thème ne sera pas ajouté à l'index s'il figure dans la directive !SANSINDEX.

Comme vous l'avez deviné, CREEAIDE affecte un numéro d'index par défaut, $7777 (30 583 en décimal) à chaque thème et la directive !INDEX modifie cette valeur. L'index est trié en primaire sur le numéro d'index et en secondaire sur le nom du thème.

Exemple

```
!THEME 1 Aide sur l'aide
!INDEX 1
...
!THEME 2 Mots réservés
!SANSINDEX
...
!THEME 3 Liste alphabétique
!THEME 4 Types numériques
!INDEX 32000
```

Le premier thème de l'index sera "Aide sur l'aide" puisque son index vaut 1.

Le deuxième thème "Mots réservés" n'apparaîtra pas dans l'index (!SANSINDEX).

Le troisième thème apparaîtra dans l'index dans l'ordre alphabétique.

Le quatrième thème reçoit un numéro très grand, ce sera le dernier thème de l'index.

!LIGNE

Cette directive permet de marquer la fin du texte en insérant un saut de ligne. Puisque CREEAIDE découpe et restructure votre texte initial, les sauts de ligne sur les écrans d'aide ne correspondront pas nécessairement à ceux du texte initial. la commande !LIGNE vous permet de contrôler ce découpage si nécessaire. CREEAIDE insère également un saut de ligne à la rencontre d'une ligne vide ou d'une ligne qui commence par un ou plusieurs espaces.

!PAGE

Cette directive permet de marquer la fin de la page courante du texte en insérant un saut de page.

Normalement, TPAIDE pagine le texte seulement à l'affichage. Vous pouvez pour améliorer la présentation placer des sauts de page où vous le souhaitez dans votre fichier d'aide.

Les commentaires du fichier texte n'apparaîtront pas à l'écran. Un commentaire commence par un point-virgule.

Exemple

```
;Ceci est un commentaire ; Il sera ignoré
```

Caractères de contrôle

Certains caractères de contrôle offrent un autre moyen pour personnaliser la présentation des écrans d'aide. Vous pouvez activer des attributs vidéo spéciaux ou spécifier des liens entre thèmes par table de références croisées.

Voici trois caractères de contrôle des attributs vidéo :

```
^A      attribut 1      Couleurs[Att1Sp]
^B      attribut 2      Couleurs[Att2Sp]
^C      attribut 3      Couleurs[Att3Sp]
```

Le tableau de couleurs mentionné lors de l'ouverture du système d'aide définit les attributs réels associés à chaque bascule.

Les caractères de contrôle peuvent apparaître n'importe où dans le texte d'aide mais sont interdits dans les directives. Ils ne sont pas affichés et ne modifient pas la longueur des lignes lors du découpage.

La première occurrence d'un caractère de contrôle affiche le texte qui suit avec la couleur associée, la deuxième occurrence du même caractère désactive cette couleur et redonne à la couleur courante la valeur

précédente. Le fonctionnement est identique à celui de la routine *EcritFacile* de TPCRT.

Pour saisir les caractères de contrôle sous l'éditeur de *Turbo Pascal,* pressez tout d'abord (CTRL)-P, maintenez enfoncée la touche (CTRL) et pressez le caractère désiré. Par exemple pour saisir ^A, pressez (CTRL)-P, (CTRL)-A.

Deux caractères de contrôle supplémentaires permettent de mettre en oeuvre des possibilités de références croisées du système d'aide. Une table de références croisées permet de passer d'un texte à un autre à l'intérieur du système d'aide. Lorsque l'aide est affichée, l'utilisateur peut positionner une barre lumineuse sur le lien de référence croisée en pressant les touches du curseur. L'utilisateur peut alors presser (ENTER) pour sélectionner le thème référencé et afficher l'aide associée et parcourir de nombreux thèmes reliés entre-eux. Un thème donné peut être lié au maximum à 50 références croisées.

Les caractères de contrôle à l'intérieur d'un texte d'aide doivent par conséquent offrir deux fonctions

- Tout d'abord, ils doivent indiquer quel thème d'aide doit être activé lorsqu'une référence croisée est sélectionnée.
- Deuxièmement , ils doivent préciser le suffixe de la référence lumineuse du texte.

TPAIDE utilise la syntaxe suivante à cet effet :

```
^Dnnnn^EText^E
```

Le caractère de contrôle ^D précède le numéro du thème référencé "nnnn" qui est exprimé en notation décimale. Le numéro de thème se termine par ^E qui est immédiatement suivi par le passage du texte à mettre en valeur. Un deuxième ^E termine le texte mis en valeur. Remarquez que le caractère ^E apparaît deux fois et le caractère ^D apparaît une seule fois. Voici un exemple de deux références croisées :

```
^D10^ELa compilation^E du texte d'aide est facile avec ^D5^ECREEAIDE^E
```

Dans le texte réel, les caractères précédés d'un caret doivent représenter des caractères de contrôle réels. Si l'utilisateur positionne la barre lumineuse sur le mot CREEAIDE et presse (ENTER), l'aide associée au thème numéro 5 s'affichera.

Le texte encadré de ^E est affiché avec deux attributs. Lorsque la référence croisée à été sélectionnée, le texte apparaît avec l'attribut couleur *Couleurs[AttrXs],* sinon il apparaît avec l'attribut *Couleurs[AttrXr].*

Le programme de démonstration SAISIE.EXE fourni avec *Turbo Professionnel* utilise ce système d'aide. Consultez comme exemple le fichier d'aide SAISIE.TXT.

Exécution de CREEAIDE

CREEAIDE est un utilitaire lancé depuis le DOS qui lit votre fichier texte et crée un fichier d'aide binaire prêt à l'emploi. Syntaxe de l'appel depuis le DOS :

```
CREEAIDE FichierTexteAide[FichierAideBinaire]
```

CREEAIDE lira le fichier texte appelé *FichierTexteAide* et créera un nouveau fichier binaire appelé *FichierAideBinaire*. Si *FichierAideBinaire* n'est pas spécifié, CREEAIDE utilisera le nom du fichier texte suivi du suffixe .HLP. CREEAIDE écrasera tout fichier existant de même nom, sans demander de confirmation. Le fichier d'entrée et le fichier de sortie doivent avoir des noms différents. Par exemple :

```
CREEAIDE DEMO.TXT DEMO.HLP
CREEAIDE DEMO.TXT
```

Ces deux commandes font la même chose.

CREEAIDE effectue deux parcours du fichier d'entrée : un pour compter les éléments fondamentaux tels que le nombre de thèmes, et un second pour formater le texte et écrire le fichier de sortie compacté.

Si CREEAIDE trouve une erreur, il s'arrêtera et affichera le type de l'erreur et le numéro de l'erreur dans votre fichier texte. CREEAIDE affichera une erreur fatale dans les cas suivants :

Problèmes d'entrée/sortie

Vérifiez que les lecteurs sont prêts et qu'il y a suffisamment de place pour le fichier de sortie.

Tableau des noms de thèmes supérieur à 64K

Utilisez des noms de thèmes plus courts ou diminuez le nombre de thèmes du fichier d'aide.

Mémoire insuffisante

CREEAIDE n'a pas assez de mémoire pour les stockages temporaires. Libérez de la mémoire en supprimant des programmes résidents ou diminuez la taille du fichier d'aide.

Index du fichier supérieur à 64K

Le nombre de thèmes dépasse 10 000. Diminuez le nombre de thèmes dans le fichier d'aide.

Pas de thème d'aide spécifié

Il n'y a pas de directive !THEME dans le fichier d'entrée.

Numéro de référence croisée incorrect

Le numéro suivant un caractère ^D ne correspond pas à un thème du fichier d'aide.

Largeur non valide, Elément non valide, Index non valide

Le numéro suivant une directive d'aide est incorrect.

Numéro de thème d'aide en double

Le même index a déjà été affecté à un thème d'aide.

L'instruction LARGEUR doit précéder le premier thème d'aide

Le directive !LARGEUR doit être la première directive du fichier d'aide.

L'instruction INDEX doit suivre l'instruction !THEME

Chaque directive !INDEX ou !SANSINDEX doit suivre la directive !THEME qui la concerne.

Directive non reconnue

La colonne 1 du fichier d'aide contient un point d'exclamation, mais le libellé qui le suit n'est pas une directive d'aide valide.

Dans certaines situations, CREEAIDE affiche un avertissement et poursuit son travail. Le message d'avertissement apparaît lorsque un passage d'un texte marqué comme référence croisée est éclaté sur plus d'une ligne de texte. Lorsque ces lignes sont affichées sur une seule page d'aide, il n'y a pas de problème. Par contre, si les lignes chevauchent deux pages d'aide, la partie du texte sur le seconde page ne sera pas adressable par l'utilisateur. Ce n'est pas un problème en vérité, mais cela peut conduire à certaines confusions. Si vous le souhaitez, modifiez le texte d'aide de telle façon que la référence croisée ne soit pas à cheval sur plusieurs lignes.

Une fois le traitement sur le fichier d'aide terminé, CREEAIDE génère certaines statistiques : nombre de thèmes, nombre de thèmes qui comportent un titre, la taille du plus grand thème (ce qui définit la taille du buffer d'aide à l'exécution), et taux de compression du fichier d'aide.

Incorporation d'un fichier d'aide dans un programme

TPAIDE peut utiliser directement le fichier binaire produit par CREEAIDE. Si le fichier d'aide est assez petit (inférieur à 64K), vous pouvez le lier à votre application.

Vous pouvez le faire avec l'utilitaire BINOBJ.EXE fourni par Borland avec *Turbo Pascal*. BINOBJ convertit le fichier d'aide binaire en fichier OBJ qui pourra être intégré à votre programme.

Voyons un exemple. Soit un fichier texte d'aide appelé MONAIDE.TXT. Vous devrez tout d'abord exécuter CREEAIDE pour créer le fichier d'aide binaire :

```
CREEAIDE MONAIDE.TXT MONAIDE.HLP
```

Puis, lancez BINOBJ pour créer le fichier OBJ.

```
BINOBJ MONAIDE.HLP MONAIDE.OBJ MONAIDE
```

Le troisième paramètre de BINOBJ spécifie le nom d'un identificateur public grâce auquel votre programme peut référencer le contenu du fichier OBJ.

Maintenant, dans votre programme d'application, insérez les lignes suivantes :

```
{$L MONAIDE.OBJ}
procedure MONAIDE; external;
```

La directive du compilateur $L indique à Turbo de lier MONAIDE.OBJ à votre programme.

La procédure MONAIDE offre à votre programme le moyen de référencer le contenu du fichier OBJ.

Finalement, pour initialiser le système d'aide, appelez *OuvreMemAide* comme suit :

```
Statut := OuvreMemAide(@MonAide,
                       ColGauche, LigneSup, LigneBas, ColsPick,
                       Couleurs, Aide);
if Statut <> 0 then
   {Erreur - Pas d'aide disponible};
```

La notation "@MonAide" passe l'adresse de la structure binaire à la procédure d'initialisation du système d'aide.

Vous pouvez alors référencer simplement le variable *Aide* envoyée par *OuvreMemAide* comme si vous référenciez celle qui est initialisée par *OuvreFicAide*.

TPAIDE : Déclarations

Type

```
TypeCouleurAide =   (AttrCdre,      {Couleur du cadre de la fenêtre}
                     AttrTe,        {Couleur normale du texte d'aide}
                     AttrEn,        {Couleur de l'entête de fenêtre}
                     AttrXs,        {Couleur pour l'élément sélectionné
                                     dans la référence croisée}
                     AttrXr,        {Couleur pour l'élément non
                                     sélectionné dans la référence
                                     croisée}
                     Att1Sp,        {Couleur pour le texte d'aide avec
                                     le caractère de contrôle ^A}
                     Att2Sp,        {Couleur pour le texte d'aide avec
                                     le caractère de contrôle ^B}
                     Att3Sp,        {Couleur pour le texte d'aide avec
                                     le caractère de contrôle ^C}
TabCouleurAide = array[TypeCouleurAide] of byte;
```

TypeCouleurAide définit les différentes couleurs utilisées dans une fenêtre d'aide. *TabCouleurAide* permet de rassembler ces couleurs en une seule structure. Voir les explications précédentes pour plus de détails sur le rôle des attributs.

```
PrtAide = ^DescAide;
```

Une variable de ce type joue le rôle d'interface entre votre programme et le système d'aide. TPAIDE initialise une variable de ce type lorsque vous ouvrez un fichier d'aide en allouant de la place sur le tas pour enregistrer les informations utilisées par le système d'aide. Voir TPAIDE.PAS pour plus de détails sur le type *DescAide*.

```
TAtype = TARien..TAIndex;
```

TAtype définit l'ensemble des actions exécutées par le système d'aide. Elle inclut des actions telles que le déplacement du curseur, la pagination, l'affichage de l'index d'aide et l'échappement. Voir les constantes ci-dessous pour plus de détails.

Constantes

```
MaxPagesParSection = 20;
MaxXrefsParSection = 50;
MaxThemes = 10000;
MaxPileAide = 19;
```

Ces constantes spécifient la capacité du système d'aide et s'expliquent d'elle-même. *MaxPileAide* définit le nombre de thèmes d'aide précédents que TPAIDE peut mémoriser. Ces thèmes précédents affichés peuvent être rappelées en pressant (ALT)-(F1) à l'intérieur du système d'aide. A l'exception de *MaxThemes*, la modification de ces constantes touche directement

l'utilisation du tas et de la pile de TPAIDE (l'espace utilisé pour les thèmes dépend du fichier d'aide lui-même et n'est pas directement touché par la valeur de *MaxThemes*). Vous pouvez augmenter ou diminuer ces valeurs et recompiler TPAIDE.

```
BasculeAttr1 = ^A;
BasculeAttr2 = ^B;
BasculeAttr3 = ^C;
MarqueurIndex = ^D;
BasculeXref = ^E;
MarqueSautLigne = ^M;
MarqueSautPage = ^L;
MarqueFinSect = #0;
```

Ces constantes sont les caractères utilisés dans le système d'aide pour le formatage du texte. Les caractères de contrôle ^A..^E ont été expliqués précédemment. ^M marque la fin de chaque ligne de texte d'aide et ^L marque un saut de page. Chaque thème d'aide se termine par un marqueur #0. Dans le fichier d'aide binaire, ces délimiteurs sont enregistrés en format compressé, aussi ne cherchez pas à les reconnaître.

```
IdAide : array[0..3] of char = 'TPH2';
```

Chaque fichier d'aide binaire commence par une suite de caractères précisant le format actuel.

```
TARien = 0;      {N'est pas une commande}
TAAlpha = 1;     {Caractère alphanumérique ignoré}
TAHaut = 2;      {Déplacement du curseur vers le haut sur la référence
                  croisée précédente}
TABas = 3;       {Déplacement du curseur vers le bas sur la référence
                  croisée suivante}
TAPgup = 4;      {Affiche la page d'aide précédente}
TAPgdn = 5;      {Affiche la page d'aide suivante}
TAGauche = 6;    {Déplacement à gauche du curseur sur la référence
                  croisée précédente}
TADroite = 7;    {Déplacement à droite du curseur sur la référence
                  croisée suivante}
TAQuitte = 8;    {Sortie du système d'aide}
TASelecte = 9;   {Sélectionne la référence croisée actuelle et affiche
                  le thème}
TAPreced = 10;   {Affiche le thème d'aide le plus récent}
TAHome = 11;     {Affiche la première page d'aide}
TAEnd = 12;      {Affiche la dernière page d'aide}
TAIndex = 13;    {Affiche l'index de tous les thèmes d'aide}
TASonde = 14;    {Contrôle la position de la souris et exécute une
                  action}
TAUtl0 = 15;     {Commande de sortie utilisateur}
....
TAUtl3 = 18;
```

Codes des commandes de sélection et de déplacement du curseur de TPAIDE. La plupart s'expliquent d'elles-même. Les quatre dernières (*TAUtl0..TAUtl3*) servent à définir des commandes de sortie utilisateur. Si l'une de ces commandes est exécutée, elle aura le même effet que si vous aviez pressé (ESC). Vous devrez alors contrôler la variable globale *NumCmdAide* et exécuter la fonction associé à cette commande. Pour plus de détails sur la manière d'implémenter des commandes utilisateur, voir l'appendice C.

```
TitreAide : string[39] = ' Thèmes ';
```

TPAIDE affiche cette chaîne en haut de la fenêtre d'aide toutes les fois qu'il affiche une liste Pick des thèmes d'aide.

```
UtiliseCadreAide : Boolean = True;
```

Initialisez cette constante à "False" avant d'appeler *OuvreFicAide* ou *OuvreMemAide* si vous ne souhaitez pas encadrer la fenêtre d'aide.

```
SuiteAide : Boolean = True;
```

Initialisez cette constante à "False" avant d'appeler *OuvreFicAide* ou *OuvreMemAide* si vous ne souhaitez pas que les messages "PgUp/PgDn" apparaissent dans le cadre de la fenêtre d'aide, lorsqu'il y a plus d'une page d'aide.

```
CacheCurseur : Boolean = True;
```

Initialisez cette constante à "False" avant d'appeler n'importe quelle fonction d'affichage d'aide si vous souhaitez que le curseur clignote et reste visible pendant l'ouverture de la fenêtre d'aide.

```
DerouleAideSouris : Boolean = True;
```

Lorsque la souris de TPAIDE est activée (en appelant *ActiveAideSouris*), cette constante vérifie si TPAIDE affiche et supporte la barre de défilement. Puisque la barre de défilement apparaît à droite dans le cadre de la fenêtre d'aide, elle n'est disponible que lorsque la fenêtre est encadrée. La barre de défilement apparaît seulement lorsque le thème d'aide courant est composé de plus d'une page.

La barre de défilement a plusieurs rôles. Dans les coins supérieur et inférieur du cadre, les flèches verticales sont affichées. TPAIDE déroule une page toutes les fois que le curseur de la souris a été positionné sur l'une de ces flèches et que le bouton gauche est pressé. TPAIDE affiche également un ascenseur à l'intérieur du cadre vertical pour indiquer la position relative de la page actuelle à l'intérieur du thème. TPAIDE affiche une nouvelle page et adapte l'ascenseur lorsque le curseur de la souris est placé à un emplacement différent dans la barre de défilement et que le bouton gauche est pressé.

Notez que vous devez initialiser la valeur de *DerouleAideSouris* avant d'appeler *OuvreFicAide* ou *OuvreMemAide* si vous voulez que les modifications soient effectives.

Variables

```
PtrToucheAide : Pointer;
```

Pointe sur une routine qui renverra la prochaine touche. Voir l'appendice C pour plus de détails.

```
NumCmdAide : TAType;
```

Dernière commande d'aide donnée à TPAIDE. Utile seulement s'il existe des commandes de sortie utilisateur qui nécessite un traitement spécial une fois que la fenêtre d'aide est fermée.

ActiveAideSouris

Déclaration

```
procedure ActiveAideSouris;
```

Objet

Active le contrôle des écrans d'aide par la souris.

Commentaire

Appelez cette routine pour activer le contrôle d'un système d'aide par la souris. La routine vérifie si la souris est installée, puis initialise un indicateur interne pour les autres routines Pick. Vous devrez appeler *ActiveAideSouris* une première fois avant d'appeler la routine *FenetreAide* de TPAIDE.

Puisque le contrôle par la souris de TPAIDE dépend de la gestion du clavier par TPSOURIS, *ActiveAideSouris* appelle aussi la routine *ActiveAttenteEvenenement* de TPSOURIS.

Vous n'aurez aucun problème si vous avez déjà appelé cette routine. N'oubliez pas cependant que tout gestionnaire d'événements de la souris activé précédemment perdra le contrôle après l'appel de *ActiveAideSouris*.

ActiveAideSouris ne modifie pas dans la fenêtre actuelle de la souris, le type curseur de la souris, le ratio Mickey-en-pixels pas plus qu'elle n'affiche le curseur de la souris. Votre programme devra initialiser ces valeurs de façon appropriée en appelant les routines de TPSOURIS. Vous pouvez utiliser un curseur de souris logiciel ou matériel.

Lorsque la souris est activée, TPAIDE met en oeuvre un certain nombre de fonctions supplémentaires pendant l'affichage des écrans d'aide. Voir la procédure *AffAide* pour plus de détails.

Si la fenêtre d'aide comporte un cadre et que la constante *DerouleSourisAide* de TPAIDE vaut "True", la souris peut aussi être utilisée pour balayer les pages de chaque thème.

Les actions mises en oeuvre lorsque les boutons de la souris sont pressés, peuvent être configurées de la même manière que les touches. Voir l'appendice C pour plus de détails.

Notez que les actions de la souris sont mises en oeuvre lorsque les boutons sont relâchés et non lorsqu'ils sont pressés. Cette technique permet une utilisation fiable des commandes qui exigent de presser simultanément les deux boutons. Vous pouvez modifier ce comportement en changeant la valeur de la constante *AttenteRelacheBouton* de TPSOURIS. Après cette modification, les actions seront effectuées dès que les boutons seront pressés.

Dans un programme résident utilisant TPAIDE, vous ne pourrez pas activer la souris avant de lancer le programme résident. Vous pouvez l'activer après le lancement du programme et la désactiver avant le retour. Tenez compte de ces informations lors de l'écriture de programmes résidents qui utilisent la souris. Voir les routines concernant la souris pour plus de détails.

Exemple

```
begin
   ClrScr;
   TpAide.DerouleAideSouris := True;
   ActiveAideSouris;

   Statut := OuvreFicAide('DEMO.HLP',      {Nom du fichier d'aide
                                            binaire}
                          10, 2,           {Coin supérieur gauche de la
                                            fenêtre d'aide}
                          22, 2,           {Ligne inférieure et
                                            colonnes de l'Index d'aide}
                          CouleursAide,    {Couleurs à utiliser}
                          Aide);           {Descripteur d'aide}
   if Statut <> 0 then
      {Erreur} ;

   TpFenetr.Ombre := True;
   TpFenetr.AttrOmbre := $70;
   repeat
      Theme := ListeAide(Aide, 2, 2, 10, 1);
      if Theme <> 0 then
         if not AffAide(Aide, Theme) then
            WriteLn('Impossible d''afficher l''aide pour le thème ',
                    Theme);
   until Theme = 0;

   FermeAide(Aide);
end.
```

Montre les modifications à apporter à l'exemple de l'introduction de TPAIDE pour supporter la souris dans un système d'aide.

Voir aussi

DesactiveAideSouris (TPAIDE) AffAide (TPAIDE)

ActiveIndexAide

Déclaration

```
procedure ActiveIndexAide;
```

Objet

Active l'index d'aide (F1) à l'intérieur de la fenêtre d'aide.

Commentaire

Cette procédure a des effets opposés à la procédure *DesactiveIndexAide* et reactive l'utilisation de la touche (F1) à l'intérieur d'une fenêtre d'aide "popup". Voir *DesactiveIndexAide* pour plus de détails.

Voir aussi

DesactiveIndexAide (TPAIDE)

AffAide

Déclaration

```
function AffAide(Aide : PtrAide, Theme : Word): Boolean;
```

Objet

Affiche l'écran d'aide pour le thème et renvoie "True" en cas de succès.

Commentaires

Routine principale d'affichage de l'aide. Etant donné une variable initialisée *Aide* et un numéro de thème correct, *AffAide* affiche la fenêtre d'aide sur l'écran et affiche la première page d'aide du thème. L'utilisateur peut alors exécuter les actions suivantes :

(PGUP)/(PGDN) Affiche la page d'aide précédente ou la page suivante.

(ESC) Sortie de la fenêtre d'aide et restauration de l'écran à son état précédent

(←)(↑)(→)(↓) Déplace la barre lumineuse dans la table de références croisées des thèmes, affichée éventuellement sur la page d'aide courante.

(ENTER) Sélectionne le thème mis en valeur dans la table de références croisées et affiche la première page d'aide pour ce thème.

(F1) Affiche un index de tous les thèmes d'aide et permet à l'utilisateur de sélectionner de nouveaux thèmes en utilisant les touches de déplacement et en appuyant sur (ENTER). Si l'utilisateur presse (ESC) à partir de l'index, la fenêtre d'aide est effacée et *AffAide* s'arrête. Si l'utilisateur presse (ALT)-(F1) depuis l'index, le thème d'aide précédent est restauré.

(ALT)-(F1) Affiche le dernier thème d'aide sélectionné. *AffAide* garde trace des 19 derniers thèmes d'aide sélectionnés dans la pile. Chaque fois qu'un autre thème est sélectionné, *AffAide* place le thème d'aide sur cette pile. Chaque fois que l'utilisateur presse (ALT)-(F1), l'élément du sommet de pile est chargé et l'aide correspondante affichée.

Lorsque la souris a été activée en appelant *ActiveAideSouris* et que la barre de déroulement a été activé en positionnant *DerouleAideSouris* à "True" (valeur par défaut), la routine *AffAide* permet d'utiliser la souris de plusieurs manières. Tout d'abord, le bouton droit de la souris correspond à la frappe de (ESC), c'est-à-dire sortie du système d'aide. Deuxièmement, la frappe simultanée des deux boutons de la souris correspond à la frappe de (F1) dans le système d'aide, c'est-à-dire que l'index d'aide est chargé.

La souris peut aussi être utilisée pour sélectionner des thèmes dans une table de références croisées. En positionnant le pointeur de la souris sur une table de références croisées et en pressant le bouton gauche vous déplacerez la barre lumineuse. En pressant une deuxième fois le bouton gauche, vous sélectionnerez le thème et afficherez un nouvel écran d'aide comme si vous pressiez la touche (ENTER).

Dans le coin supérieur gauche de la fenêtre d'aide se trouve un petit motif en forme de losange. En positionnant le pointeur de la souris sur celui-ci et en pressant le bouton gauche, vous réafficherez le dernier thème sélectionné comme si vous pressiez (ALT)-(F1).

Le coté droit de la fenêtre d'aide contient une barre de défilement (si la fenêtre d'aide ne comporte pas de cadre ou s'il n'y a qu'une seule page de thèmes d'aide, cette barre n'apparaîtra pas). Positionnez le pointeur de la souris sur les flèches en haut ou en bas de la barre de défilement et appuyez sur le bouton gauche pour que le routine *AffAide* affiche la page d'aide précédente ou la page suivante respectivement. Dans la partie centrale de la barre de sélection, TPAIDE affiche un caractère appelé "ascenseur". La position du curseur indique la positon relative de la page d'aide courante à l'intérieur du thème. Positionnez le pointeur de la souris au-dessous ou au-dessus de l'ascenseur et pressez le bouton gauche pour que TPAIDE affiche une nouvelle page d'aide en fonction de la position du pointeur de la souris.

Toute commande peut être désactivée ou réaffectée à de nouvelles touches. Voir l'appendice C pour plus de détails.

Si une erreur se produit lors de l'affichage de l'aide, *AffAide* renvoie "False". Les causes d'erreurs peuvent être "Mémoire insuffisante" (pour un buffer de fenêtre), "Erreur disque" (longueur du fichier d'aide non valide) ou bien "Numéro de thème d'aide non valide" (0 ou numéro inexistant). De plus, lorsque l'utilisateur appelle l'index d'aide, *AffAide* doit allouer des buffers supplémentaires pour les noms de thèmes. S'il n'y a pas assez de place sur le tas, *AffAide* ne fera que refuser l'accès à l'index d'aide et poursuivra son traitement.

L'utilisation des fenêtres dans *AffAide* dépend de l'unité TPFENETR. Par conséquent, vous pouvez tirer profit des modes spéciaux de TPFENETR tels que les ombres et les explosions pour adapter la présentation de vos écrans d'aide.

Vous pouvez centraliser les appels à *AffAide* en tirant profit des gestionnaires de clavier personnalisables de *Turbo Professionnel.*

Pour cela, vous devez procéder comme suit :

```
const
   ToucheAide : Word = $3800; {Touche F1}
var
   ThemeActuel : Word;
   Aide : PtrAide;

{$F+}
function LitToucheAide : Word;
var
   TA : Word;
begin
   repeat
      {Lit la prochaine touche frappée au clavier}
      TA := TpCrt.LitMotTouche;
      if TA = ToucheAide then
         {Demande d'aide}
         if not AffAide(Aide, ThemeActuel) then
            Write(^G);
   until TA <> ToucheAide;
   LitToucheAide := TA;
end;
{$F-}
```

Vous devez faire pointer les gestionnaires de clavier de toutes les autres unités (telles que TPEDIT, TPPICK, TPSAISIE) sur cette routine. Voir l'appendice C pour plus de détails. Votre programme devra maintenir la variable *ThemeActuel* à jour par rapport au thème d'aide. Quel que soit le mode d'entrée clavier choisi, l'utilisateur pourra presser la touche d'aide et l'aide pour le thème courant sera affichée.

Exemple

```
if not AffAide(Aide, 1) then
   Write(^G);
```

Le numéro de thème peut être une constante, une variable globale contenant le numéro de thème courant, ou une valeur calculée en fonction du nom du thème ou de l'écran.

Voir aussi

DesactiveIndexAide (TPAIDE) ActiveIndexAide (TPAIDE)

AffAidePreced

Déclaration

```
function AffAidePreced(Aide : PtrAide): Boolean;
```

Objet

Affiche l'écran d'aide pour le dernier thème sélectionné et renvoie "True" en cas de succès.

Commentaires

Cette routine permet à votre programme d'accéder directement à la pile de TPAIDE qui contient les thèmes d'aide précédemment affichés. Lorsque vous appelez *AffAidePreced*, elle affiche le thème précédent et le "dépile". Si la pile est vide ou si une autre erreur se produit, *AffAidePreced* renvoie "False".

Toutes les fonctionnalités de *AffAide* sont disponibles pendant l'exécution de *AffAidePreced*.

Exemple

```
if not AffAidePreced(Aide) then
   Write(^G);
```

Voir aussi

AffAide (TPAIDE)

AideEcran

Déclaration

```
function AideEcran(Aide : PtrAide; PosX, PosY : byte;
                   RechArriere : Boolean; FoncRech : Pointer): Word;
```

Objet

Renvoie le numéro du thème dont le nom correspond au contenu de l'écran à la position (*PosX, PosY*) et 0 sinon.

Commentaires

Vous utiliserez cette fonction pour offrir un système d'aide basé sur une analyse de l'écran. Par exemple, supposez qu'un programmeur désire une information sur une procédure particulière et que le curseur soit positionné sur le nom de la procédure. *AideEcran* parcourra le mot le plus près du curseur, recherchera dans le système d'aide un thème correspondant à ce nom et renverra éventuellement le numéro du thème.

Le paramètre *Aide* passé à *AideEcran* doit avoir été initialisé par un appel à *OuvreFicAide* ou *OuvreMemAide*. *PosX* et *PosY* sont les coordonnées absolues à partir desquelles *AideEcran* effectuera son analyse. Si *RechArriere* vaut "True", *AideEcran* recherchera à partir de la position *PosX* vers la gauche en ignorant tout caractère qui ne fait pas partie d'un mot. Considérons l'exemple suivant :

```
Il a beaucoup plu lors de notre voyage en Espagne.      _
```

Si *RechArriere* vaut "False", *AideEcran* notera que le curseur n'est pas à l'intérieur d'un mot et s'arrêtera immédiatement sans analyse supplémentaire. Si *RechArriere* vaut "True", *AideEcran* balayera l'écran vers l'arrière en commençant juste à la gauche du curseur, sautera les espaces et le point, analysera le mot Espagne de la fin de ligne et recherchera alors le thème "Espagne".

Lors de l'analyse des mots sur l'écran, *AideEcran* considère que les caractères suivants sont des caractères valides dans un mot :

```
'A'..'Z', '_', '0'..'9'
```

FoncRech spécifie l'adresse d'une fonction compilée avec le modèle **FAR** qui indique si le mot analysé sur l'écran correspond à un nom de thème d'aide donné. Si vous passez la valeur nil dans ce paramètre, *AideEcran* utilise sa fonction de comparaison par défaut qui suppose que le mot analysé et le nom de thème sont rigoureusement égaux (sans tenir compte des majuscules).

Vous pouvez écrire une routine qui offre une recherche moins rigoureuse pour rechercher le mot analysé sur l'écran n'importe où à l'intérieur d'un nom de thème d'aide. Votre fonction de comparaison doit être déclarée selon la syntaxe suivante :

```
{$F+}
function Correspond(NomTheme, NomChn : string) : Boolean;
{$F-}
```

Cette fonction renverra "True" si les deux chaînes sont égales. Le premier paramètre est le thème en cours de comparaison, et le second paramètre est le mot analysé sur l'écran qui sera le même pour chaque appel.

Lorsqu'il y a correspondance, *AideEcran* renvoie l'index du thème d'aide correspondant, sinon *AideEcran* renvoie la valeur 0. Tout index différent de 0 renvoyé par *AideEcran*, peut être utilisé comme paramètre de *AffAide* afin d'afficher l'aide correspondant à ce thème.

Exemple

```
{$F+}
function Correspond(NomTheme, NomChn : string) : Boolean;
begin
   {Correspond si le mot est contenu dans le nom du thème}
   Correspond := Pos(ChnMajusc(NomChn), ChnMajusc(NomTheme)) <> 0;
end;
{$F-}

begin
   Theme := AideEcran(Aide, CoordXAbs, CoordYAbs, True, @Correspond;
end.
```

AideEcran affiche le mot le plus près de la position du curseur sur l'écran et renvoie le thème dont le nom contient ce mot.

Voir aussi

ChercheAide (TPAIDE)

ChercheAide

Déclaration

```
function ChercheAide(Aide : PtrAide; Nom : string;
                     FoncRech : Pointer): word;
```

Objet

Envoie le numéro du thème d'aide pour le nom spécifié et 0 s'il n'est pas trouvé.

Commentaires

ChercheAide recherche dans les noms de thèmes du système d'aide le nom correspondant au paramètre *Nom*. Par défaut, la recherche se fait sans tenir compte des majuscules ou des minuscules. La recherche sera satisfaite s'il y a égalité stricte entre le nom spécifié et le thème courant. Le paramètre *Aide* de *ChercheAide* peut avoir été initialisé en appelant *OuvreFicAide* ou *OuvreMemAide*.

Le mode de recherche par défaut est activé lorsque le paramètre *FoncRech* contient nil. Si ce paramètre contient l'adresse d'une autre fonction, la recherche se fera en utilisant cette fonction de recherche. La fonction dont l'adresse est passée en paramètre doit être compilée avec le modèle **FAR** et elle doit correspondre à la définition suivante :

```
{$F+}
function Correspond(NomTheme, Nom : string) : Boolean;
{$F-}
```

Cette fonction renverra "True" si les deux chaînes sont égales. Le premier paramètre est le thème en cours de comparaison et le second paramètre est le paramètre passé à *ChercheAide* qui sera le même à chaque appel.

Lorsqu'il y a correspondance, *ChercheAide* renvoie l'index du thème trouvé, sinon *ChercheAide* renvoie 0. N'importe quelle valeur différente de 0 renvoyée par *ChercheAide* peut être utilisée comme paramètre de *AffAide* afin d'afficher l'aide sur ce thème.

Exemple

```
{$F+}
function Correspond(NomTheme, Nom : string) : Boolean;
begin
  {Correspondance si le thème commence par le nom recherché}
  Correspond:=Pos(ChnMajusc(Nom), ChnMajusc(EffBlancs(NomTheme))) = 1;
end;
{$F-}

begin
  Theme := ChercheAide(Aide, 'COMPIL', @Correspond);
end.
```

CherchеAide renvoie l'index du thème dont le nom commence par 'Compil' ; c'est-à-dire par exemple les thèmes 'Compilation', 'Compilateur' et ainsi de suite. Remarquez que la fonction de recherche définie par l'utilisateur est forcée pour une compilation **FAR** et que chaque chaîne est mise en majuscules, les blancs sont éliminés du thème d'aide avant d'exécuter la comparaison.

Voir aussi

AideEcran (TPAIDE)

DesactiveAideSouris

Déclaration

```
procedure DesactiveAideSouris;
```

Objet

Désactive le contrôle des écrans d'aide par la souris.

Commentaires

Cette routine annule les effets de *ActiveAideSouris*. Dans de nombreux programmes, vous n'aurez pas à appeler *DesactiveAideSouris* puisque ses fonctions essentielles sont gérées automatiquement par une routine de sortie de *Turbo Pascal*. Dans certains cas, particulièrement pour les programmes résidents, vous devrez appeler *DesactiveAideSouris* afin de désactiver temporairement les fonctions de la souris.

Notez que *DesactiveAideSouris* ne cache pas le curseur de la souris. Vous pouvez appeler *CacheSouris* pour cela.

Exemple

```
{Activation TSR}
...
ActiveAideSouris;
...
if AffAide(...) then begin
   ...
end;
DesactiveAideSouris;
...
{Désactivation TSR}
```

Voir aussi

ActiveAideSouris (TPAIDE)

DesactiveIndexAide

Déclaration

```
procedure DesactiveIndexAide;
```

Objet

Désactive l'index d'aide (F1) à l'intérieur de la fenêtre d'aide.

Commentaires

Normalement lorsqu'une fenêtre d'aide est activée et que l'utilisateur presse (F1), TPAIDE affiche automatiquement l'index maître de tous les thèmes d'aide. A partir de là, l'utilisateur peut sélectionner n'importe quel thème indexé et sélectionner l'aide pour cet élément. L'appel de *DesactiveIndexAide* dans un programme désactive l'utilisation de (F1) à l'intérieur de la fenêtre d'aide "popup". L'index d'aide peut être réactivé en appelant *ActiveIndexAide.*

DesativeIndexAide est un cas spécial de modification dynamique de l'interface clavier de l'unité TPAIDE. Voir l'appendice C pour plus de détails.

Exemple

```
Statut := OuvreFicAide('DEMO.HLP', 10, 2, 22, 2, CouleursAide, Aide);
DesactiveIndexAide;
if AffAide(Aide, 1) then ;
ActiveIndexAide;
if AffAide(Aide, 1) then ;
```

Le premier appel à *AffAide* ne permet pas d'utiliser la touche (F1) à l'intérieur de la fenêtre "popup". Le second appel le permet.

Voir aussi

ActiveIndexAide (TPAIDE)

FermeAide

Déclaration

```
Procedure FermeAide(var Aide : PtrAide);
```

Objet

Ferme la fichier d'aide et/ou libère le buffer.

Commentaires

FermeAide ferme le fichier d'aide s'il a été ouvert et libère tous les buffers pour le système d'aide. Le paramètre *Aide* doit avoir avoir été initialisé précédemment pas *OuvreFicAide* ou *OuvreMemAide*. Après un appel à *FermeAide, Aide* contiendra nil.

Exemple

```
var
   Aide : PtrAide;
   ...
   Statut := OuvreFicAide('DEMO.HLP', 10, 2, 22, 2, CouleursAide,
Aide);
   FermeAide(Aide);
```

Ouvre et ferme immédiatement le fichier d'aide DEMO.HLP.

Voir aussi

OuvreFicAide (TPAIDE) OuvreMemAide (TPAIDE)

InitPosAide

Déclaration

```
Procedure InitPosAide(Aide : PtrAide; XBas, YBas, YHaut : Byte);
```

Objet

Modifie la position et la hauteur de la fenêtre d'aide une fois le descripteur d'aide initialisé.

Commentaires

Lorsque vous ouvrez un fichier d'aide, vous spécifiez la position à laquelle la fenêtre d'aide apparaîtra. Cette procédure vous permet de modifier cette position ultérieurement. Notez que la routine ne peut pas être appelée lorsque la fenêtre d'aide est en cours d'affichage.

Exemple

```
var
   Aide : PtrAide;
   Statut : Word;
begin
   Statut := OuvreFicAide('DEMO.HLP', 10, 2, 22, 2, CouleursAide,
                          Aide);
   ...
   InitPosAide(Aide, 15, 5, 22);
end.
```

L'appel à *InitPosAide* diminue la taille de la fenêtre.

ListeAide

Déclaration

```
function ListeAide(Aide : PtrAide; XBas, YBas, YHaut : Byte;
                   ColsPick : Byte): Word;
```

Objet

Affiche l'index d'aide et renvoie le numéro du thème sélectionné, 0 sinon.

Commentaires

Appelez cette routine pour afficher la liste Pick des thèmes d'aide disponibles. Cette fonction est identique à un index d'aide qui apparaît lorsque l'utilisateur presse (F1) dans une fenêtre d'aide avec deux exceptions. La première est que les coordonnées de la fenêtre de *ListeAide* sont initialisées indépendamment par les paramètres que vous lui passez, contrairement à la liste Pick interne dont les coordonnées correspondent à celles de la fenêtre elle-même. Deuxièmement, *ListeAide* renvoie un numéro d'index de thèmes au lieu d'afficher immédiatement l'aide.

Le paramètre *Aide* doit avoir été initialisé en appelant *OuvreFicAide* ou *OuvreMemAide*. *XBas* et *YBas* spécifient le coin supérieur gauche de la fenêtre Pick, y compris le cadre s'il existe (l'initialisation de la constante globale *UtiliseCadreAide* indique si la fenêtre Pick comporte un cadre). *YHaut* est la ligne inférieure de la fenêtre Pick adaptée conformément aux règles de la routine *FenPick*. *ColsPick* est le nombre de colonnes de l'élément d'aide à afficher. *ListeAide* doit allouer deux buffers temporaires sur le tas, un buffer contenant tous les noms des thèmes d'aide qui peut avoir une taille de 64 K en fonction de la taille du système d'aide. L'autre contient une liste des thèmes triés qui indique l'ordre dans lequel les index d'aide apparaîtront et utilise deux octets pour chaque thème d'aide. Si la place sur le tas est insuffisante, *ListeAide* s'arrêtera immédiatement est renverra un numéro de thème égal à 0.

ListeAide affiche ensuite la liste Pick et permet à l'utilisateur de sélectionner un thème. Si l'utilisateur presse (ESC), *ListeAide* renvoie le numéro de thème 0 sinon elle renvoie le numéro du thème sélectionné.

Si la souris est activée pour TPAIDE, *ListeAide* active temporairement le support de la souris pour TPPICK.

Exemple

```
TpAide.UtiliseCadreAide := False;
TpPick.RechPick := ChaineRechPick;
TpFenetr.Explosion := True;
Theme := ListeAide(Aide, 2, 2, 10, 1);
if Theme <> 0 then
   if not AffAide(Aide, Theme) then
      WriteLn('Impossible d''afficher l''aide pour le thème ', Theme);
```

Ceci vous montre comment utiliser les différents modes des unités dont dépend TPAIDE pour adapter la présentation de la fenêtre Pick. Cette fenêtre particulière ne comportera pas de cadre, supportera les recherches de chaînes incrémentales et explosera au premier affichage. Son coin supérieur gauche aura pour coordonnées (2, 2) et elle affichera un maximum de 9 lignes par thème.

Voir aussi

FenPick (TPPICK)

AffAide (TPAIDE)

OuvreFicAide

Déclaration

```
function OuvreFicAide(NomFicAide : String;
                      XBas, YBas, YHaut, ColsPick : Byte;
                      Couleurs : TabCouleurAide;
                      var Aide : PtrAide): word;
```

Objet

Recherche, ouvre le fichier d'aide et, renvoie un descripteur d'aide initialisé et un code d'erreur.

Commentaires

OuvreFicAide recherche tout d'abord le fichier d'aide spécifié comme premier paramètre. Pour cela elle utilise la fonction *ExisteSurChemin* de l'unité TPDOS. Si *NomFicAide* contient une spécification de chemin, seul le répertoire de ce chemin est contrôlé. Si *NomFicAide* est un nom de fichier brut, *ExisteSurChemin* cherchera d'abord dans le répertoire courant puis, dans le répertoire dans lequel le programme lui-même a été chargé (sous DOS 3.0 ou plus) et finalement dans tous les répertoires du chemin DOS. Si le fichier n'est pas trouvé, *OuvreFicAide* renverra une erreur.

Ensuite, *OuvreFicAide* lit l'entête du fichier et vérifie que le format du fichier est correct. Sinon la fonction s'arrête et renvoie une erreur.

OuvreFicAide alloue ensuite un buffer suffisamment grand pour contenir le plus grand thème d'aide une fois qu'il a été décompacté. S'il n'y a pas assez de place sur le tas, la fonction s'arrête et renvoie une erreur.

Finalement, *OuvreFicAide* valide le nombre de colonnes Pick et diminue la valeur demandée si les noms de thèmes sont trop grands pour se placer dans la fenêtre d'aide. Souvenez-vous que les colonnes Pick sont utilisées seulement lorsque la touche F1 est pressée à l'intérieur d'une fenêtre d'aide "popup" pour afficher l'index d'aide maître.

OuvreFicAide enregistre les nombreux paramètres dans l'enregistrement pointé par la variable *Aide*. En plus des coordonnées de la fenêtre d'aide et des couleurs, elle enregistre aussi la valeur courante de *UtiliseCadreAide*, *SuiteAide* et *DerouleAideSouris* qui sont des constantes globales typées décrites précédemment.

OuvreFicAide laisse le fichier d'aide physique ouvert et mobilise un gestionnaire de fichiers du DOS. Sauf pour un petit entête, aucune partie du fichier d'aide n'est lue tant qu'il n'y a pas de demande pour afficher un thème d'aide. N'appelez pas *OuvreFicAide* plus d'une fois pour chaque système d'aide à moins que le système d'aide ait été fermé par un appel à *FermeAide*.

OuvreFicAide renvoie le code d'erreur d'exécution de *Turbo Pascal* comme résultat de la fonction. Voici la liste des codes d'erreur les plus fréquents :

0 Succès

2 Fichier non trouvé ou plus d'identificateur de fichier

100 Fin de fichier inattendue en cours de lecture du fichier d'aide

106 Format de fichier non valide

203 Place insuffisante sur le tas pour les buffers d'aide

Pour les petits fichiers d'aide, il sera parfois plus pratique de placer les données d'aide de façon permanente dans vos programmes. Voir *OuvreMemAide* pour plus de détails.

Exemple

```
Statut := OuvreFicAide('C:\MONAPP\DEMO.HLP', 10, 2, 22, 2,
                       CouleursAide, Aide);
```

OuvreFicAide recherche le fichier d'aide dans le répertoire C:\MONAPP. Voir en début de section le programme complet qui appelle *OuvreFicAide*.

Voir aussi

OuvreMemAide (TPAIDE)

OuvreMemAide

Déclaration

```
function OuvreMemAide(PtrA : Pointer;
                XBas, YBas, YHaut, ColsPick : Byte;
                Couleurs : TabCouleurAide;
                var Aide : PtrAide) : word;
```

Objet

Initialise le descripteur d'aide pour un fichier d'aide interne à un programme et renvoie un code d'erreur.

Commentaires

OuvreMemAide est un autre moyen d'initialiser un système d'aide. Contrairement à *OuvreFicAide* qui laisse le fichier d'aide ouvert et oblige le système d'aide à lire le disque chaque fois qu'un thème d'aide est affiché, *OuvreMemAide* initialise simplement certains pointeurs vers l'information qui est déjà en mémoire. En conséquence, le temps de réponse du système d'aide est plus rapide avec *OuvreMemAide* et l'utilisateur n'a pas à se soucier de l'utilisation d'un fichier d'aide séparé. D'un autre côté, la taille du fichier EXE d'une application augmentera lorsque la méthode *OuvreMemAide* est utilisée.

Pour utiliser *OuvreMemAide*, vous devez placer le fichier d'aide dans votre programme. Voir la section "Incorporation d'un fichier d'aide dans un programme" décrite précédemment dans ce chapitre pour plus de détails.

Ensuite, vous devrez simplement référencer l'adresse de la procédure d'aide en utilisant l'opérateur '@' de *Turbo Pascal*.

```
Status := OuvreMemAide(@MonAide, XBas, YBas, YHaut, ColsPick, Couleurs,
Aide);
```

Puisque la procédure virtuelle associé à un fichier d'aide interne n'est jamais réellement appelée, il n'est pas nécessaire qu'elle soit compilée avec le modèle **NEAR** ou **FAR**.

Comme *OuvreFicAide*, *OuvreMemAide* renvoie un code d'erreur, mais l'erreur "Fichier non trouvé" ne pourra jamais se produire. Comme *OuvreFicAide*, *OuvreMemAide* doit allouer un buffer sur le tas pour contenir le texte d'aide décompacté le plus grand.

Exemple

```
program DemoAide;
uses
   Dos, TpDos, TpChaine, TpCrt, TpFenetr, TpCmd, TpPick, TpAide;
const
   CouleursAide : TabCouleurAide =
                          ($07, $07, $0F, $70, $09, $70, $0F, $01);
var
   Aide : PtrAide;
   Statut : Word;
   Theme : Word;

{$L AIDEDEMO.OBJ}
procedure AideDemo; external;

begin {Programme principal}
   ClrScr;
   Statut := OuvreMemAide(@AideDemo, 10, 2, 22, 2, CouleursAide,
                          Aide);
   if Statut <> 0 then begin
      WriteLn('Erreur d''initialisation d''aide ', Statut);
      Halt;
   end;

   repeat
      {Choix d'un thème dans l'index d'aide}
      Theme := ListeAide(Aide, 2, 2, 10, 1);
      if Theme <> 0 then
         {Affiche l'aide concernant ce thème}
         if not AffAide(Aide, Theme) then
            WriteLn('Impossible d''afficher l''aide pour le thème ',
                    Theme);
   until Theme = 0;

   {Libère la mémoire}
   FermeAide(Aide);
end.
```

Exécute les mêmes fonctions que l'exemple décrit en début de section mais utilise une aide interne plutôt qu'un fichier d'aide.

Voir aussi

OuvreFicAide (TPAIDE)

TPSOURIS : Routines de gestion de la souris

En écrivant TPSOURIS, nous visions trois objectifs. Premièrement, nous voulions fournir des routines performantes pour programmer la souris en mode texte, d'une part parce que les routines existantes étaient inadaptées, d'autre part parce que les autres routines orientées écran de *Turbo Professionnel* sont prévues pour fonctionner en mode texte. Deuxièmement, nous voulions offrir une interface très conviviale dispensant autant que possible le programmeur de la gestion du pilote de la souris. Troisièmement, nous voulions offrir un ensemble de routines réutilisables dans les autres unités de *Turbo Professionnel* pour décharger le programmeur au maximum.

Démarrage de TPSOURIS

Lorsque vous commencerez à travailler avec TPSOURIS, vous serez heureux d'apprendre que nous avons construit TPSOURIS sur le modèle de TPCRT, une unité avec laquelle vous êtes probablement déjà familiarisé. Outre le gain de temps sur l'apprentissage de TPSOURIS, vous découvrez d'autres avantages. Voici la liste des correspondances entre les deux unités :

TPSOURIS	Equivalence	TPCRT
SourisPressee		KeyPressed
MotToucheSouris		LitMotTouche
InitSouris		ReinitVideo
SourisX		WhereX
SourisY		WhereY
SourisXY		WhereXY
SourisEnXY		GotoXY
FenSouris		Window
CursSourisNormal		CursNormal
CursSourisPlat		CursPlat
CursSourisBloc		CursBloc
CursSourisCache		CursCache
LitPageSouris		PageActuelle (variable globale)
InitPageSouris		InitPageVisible

A part quelques exceptions, la plupart des autres routines de TPSOURIS seront moins indispensables. Si vous avez seulement survolé TPCRT, vous aurez quelques petites difficultés à vous familiariser avec les outils de base fournis par TPSOURIS.

Souvenez-vous que la plupart des routines de cette unité ne pourront être appelées que si la variable globale *SourisInstallee* vaut "True". Cette règle n'est pas strictement obligatoire mais vous risquez d'obtenir des résultats imprévisibles si vous ne la respectez pas. Les seules véritables exceptions

sont *InitSouris* (routine utilisée pour positionner *SourisInstallee*), *SourisPressee* (renvoie toujours "False" si la souris n'est pas installée), *LitToucheOuBouton* (renvoie toujours une touche si *SourisInstallee* vaut "False") et *TailleBuffSouris* (renvoie toujours 0 s'il n'y a pas de souris). Bien que certaines routines puissent fonctionner sans problème si *SourisInstallee* vaut "False", un programme bien construit ne devrait appeler que les routines citées précédemment sauf si *SourisInstallee* vaut "True".

Support de la souris dans les autres unités

Comme nous l'avons évoqué, cinq autres unités de *Turbo Professionnel* peuvent utiliser les routines de l'unité TPSOURIS : TPEDIT, TPMENU, TPPICK, TPAIDE et TPSAISIE (TPDIR également si la souris est activée par TPPICK). Dans chaque cas , vous pouvez activer cette fonction en appelant une seule routine :

```
ActiveEditSouris     (TPEDIT)
ActiveMenuSouris     (TPMENU)
ActivePickSouris     (TPPICK)
ActiveAideSouris     (TPAIDE)
ActiveSaisieSouris   (TPSAISIE)
```

Dans la plupart des cas, vous devrez simplement spécifier le type de pointeur souris que vous désirez utiliser avant d'appeler ces routines (voir les explications du pointeur de la souris ci-dessous). Vous pouvez visualiser également le pointeur de la souris en appelant *AffSouris* mais ce n'est pas toujours nécessaire. Une fois que le support de la souris est activé dans ces unités, les routines pourront utiliser le pointeur de la souris en toute sécurité lorsque cela sera nécessaire et le restaurer à son état initial à la sortie.

Bien que vous puissiez souhaiter mettre en oeuvre d'autres utilisations de la souris dans votre propre code, il est actuellement possible d'écrire des programmes qui utilisent effectivement la souris en ajoutant simplement quelques appels de procédures en tête du programme. Si vous examinez le programme de démonstration GESTEMPS.PAS et VBROWSE.PAS, et regardez tout particulièrement les sections de codes commençant par "{$IFDEF UseMouse}", vous constaterez qu'il y a vraiment peu de choses à faire pour utiliser la souris dans les programmes.

Le pointeur de souris

Dans le paragraphe précédent, nous avons indiqué que vous auriez à choisir le type de pointeur de souris que vous voulez utiliser. Les pointeurs de souris peuvent être mis en oeuvre de deux manières : hard ou soft. Lorsque vous choisissez un pointeur de souris hardware, le pilote de la souris déplace simplement le pointeur réel dans l'écran en réponse au mouvement de la souris (dans ce qui suit, nous appellerons le pointeur réel : "pointeur

système"). Lorsque vous choisissez un pointeur software, le pilote de la souris n'utilise pas le curseur système et manipule à la place des caractères sur l'écran (modifie temporairement les couleurs et/ou les recouvre avec d'autres caractères) afin d'indiquer la position du pointeur de la souris. Lorsque la souris est déplacée, le caractère sous le pointeur de souris est restauré et un autre emplacement de l'écran est modifié.

Si vous éprouvez quelques difficultés à comprendre cela, nous vous conseillons d'essayer les programmes de la disquette DEMO avant d'aller plus loin : CREEMENU et REF.

CREEMENU utilise le pointeur de souris software (qui ressemble à un losange) car il utilise le pointeur système pour d'autres fonctions. REF utilise un pointeur de souris hardware (un curseur clignotant) que nous expliquerons plus tard.

En général, dans les programmes où cela est possible, il est préférable d'utiliser le pointeur de souris hardware. Malheureusement, ces programmes sont rares. La plupart des programmes qui peuvent utiliser la souris utilisent également le curseur système et ne sont pas à même de le déplacer chaque fois que l'utilisateur déplace la souris. TPSOURIS vous permet de choisir plusieurs formes de pointeur. Vous pouvez utiliser la routine *CursSourisHard* qui vous permet de spécifier les lignes de début et de fin du curseur, ou vous pouvez utiliser l'une des routines de plus haut niveau : *CursSourisNormal*, *CursSourisPlat*, ou *CursSourisBloc*.

Vous utiliserez un pointeur software seulement dans les cas où vous ne pouvez utiliser un pointeur hardware mais aussi dans les cas où vous voulez produire des effets spéciaux impossibles avec un pointeur hardware. Le pointeur de souris par défaut est un bloc en inverse vidéo, non clignotant et transparent dont les couleurs se modifient lors des déplacements afin de garantir un contraste suffisant avec les caractères qu'il recouvre. Comment cela est-il possible ?

Lorsque vous sélectionnez un pointeur software (en utilisant *CursSourisSoft*), vous devez spécifier un "masque d'écran" et "un masque de curseur". Le moyen le plus rapide pour comprendre la manière dont sont utilisées ces valeurs est d'étudier des exemples. Supposons que le pointeur de souris doive être déplacé vers le haut d'un caractère "A" (ASCII #$41) qui est affiché en blanc sur noir (attribut vidéo $0F). En mémoire vidéo, ce caractère est représenté comme un mot dont l'attribut est l'octet de poids fort ($0F41). Etant donné le masque d'écran par défaut de $FFFF (*MasqueEcranParDefaut*) et le masque de curseur par défaut de $7700 (*MasqueCurseurParDefaut*), le pilote de la souris créera le pointeur de souris en effectuant un ET logique entre la valeur du caractère et le masque d'écran.

```
$0F41 AND $FFFF = $0F41  <-- pas de changement
```

et un OU logique du résultat avec le masque du curseur :

```
$0F41 XOR $7700 = $7841  <-- attribut modifié, caractère inchangé
```

Ceci affichera la lettre "A" en inversion vidéo (gris foncé sur gris clair). Maintenant, supposons que vous vouliez obtenir un effet légèrement différent, des couleurs variables avec un pointeur de souris en forme de losange (ASCII #$04). Dans ce cas, vous utiliserez un masque d'écran de $FF00 et un masque de curseur de $7704 :

```
$0F41 AND $FF00 = $FF00  <-- attribut inchangé, caractère effacé
$0F00 XOR $7704 = $7804  <-- attribut modifié, nouveau caractère
                             dans l'octet de poids faible
```

Finalement, supposons que vous vouliez définir un pointeur de souris qui soit le même quelque soit sa position sur l'écran. Vous voulez toujours qu'il apparaisse sous forme d'une note de musique (ASCII #$14) affichée en blanc sur rouge ($4F). Dans ce cas, vous utiliserez un masque d'écran de $0000 et un masque curseur de $4F14 :

```
$0F41 AND $0000 = $0000  <-- attribut, caractère effacé
$0000 XOR $4F14 = $4F14  <-- attribut et caractère nouveaux
```

Lecture des boutons de la souris

Lorsque vous commencerez à mettre en oeuvre la souris dans vos programmes, vous verrez que cela est très facile si vous disposez de certaines données. Dans la plupart des cas, vous devrez savoir : où se trouve la souris et quels boutons de la souris ont été pressés. Dans cette section, nous développerons l'utilisation des boutons de la souris.

Comme nous l'avons évoqué précédemment, TPSOURIS est construite sur le modèle TPCRT. Aussi, vous ne serez pas surpris de constater que les fonctions principales de TPSOURIS sont appelées *SourisPressee* et *MotToucheSouris* qui sont aux boutons de la souris ce que *KeyPressed* et *LitMotTouche* sont au clavier. *SourisPressee* vous indique si un bouton de la souris (ou une combinaison de boutons) a été pressé et *MotToucheSouris* vous renvoie la valeur de la touche qui a été pressée. Avant de les mettre en oeuvre, il y a deux choses que vous devrez savoir. Premièrement, ces fonctions seront inopérantes tant que *ActiveAttenteEvenement* n'a pas été appelée. Nous expliquerons pourquoi dans le paragraphe "Gestionnaire d'événements de la souris". Deuxièmement, *MotToucheSouris* renvoie ce que appelons "un pseudo-code de balayage", c'est-à-dire une valeur de mot qui ressemble au code de balayage renvoyé par *LitMotTouche* :

```
SourisGauche = $EF00;          {Bouton gauche}
SourisDroit = $EE00;           {Bouton droit}
SourisLes2 = $ED00;            {Les deux boutons}
SourisCentral = $EC00;         {Bouton central}
```

```
SourisGaucheCentral = $EB00;        {Bouton gauche et bouton central}
SourisDroitCentral = $EA00;         {Bouton droit et bouton central}
SourisLes3 = $E900;                 {Les trois boutons}
```

Comme vous le voyez, l'octet de poids faible est toujours à zéro, et l'octet de poids fort indique quelle combinaison de boutons a été pressée. Ce système permet d'ajouter le support de la souris aux programmes qui sont déjà configurés pour des entrées clavier (Remarquez que ces pseudo-codes de balayage sont affichés par REF, votre programme résident d'affichage rapide de la table des caractères).

La question qui se pose est "ne serait-il pas plus simple qu'une fonction renvoie une touche ou un bouton de souris ? ". Oui en vérité, et la fonction est *LitToucheOuBouton*. Cette routine est simplement une boucle d'attente de la frappe d'une touche ou d'un bouton de souris qui renvoie ensuite le code de balayage de la touche si une touche a été pressée en premier ou le pseudo-code de balayage pour le bouton de la souris si un bouton a été pressé en premier (elle renvoie aussi, via les variables globales *MotToucheSourisX* et *MotToucheSourisY*, l'emplacement du pointeur de la souris au moment où la touche a été pressée). Votre programme utilise actuellement *LitMotTouche* pour gérer des entrées clavier. Vous avez simplement à remplacer l'appel à *LitMotTouche* par un appel à *LitToucheOuBouton* pour ajouter le code de gestion des boutons.

Il existe d'autres routines dans TPSOURIS qui renvoient des informations sur les boutons de la souris mais notre expérience nous a prouvé que celles que nous venons de voir sont les plus utiles. Pourquoi ? Parce que les autres routines peuvent simplement vous indiquer quels boutons ont été pressés à n'importe quel moment. Elles ne garantissent pas que ces boutons ont aussi été libérés. *MotToucheSouris* et *LitToucheOuBouton* le garantissent en supposant que la variable globale *AttenteRelacheBouton* ait la valeur "True". Pour apprécier l'importance de l'attente de la libération d'un bouton, essayez d'exécuter vos programmes en utilisant *LitToucheOuBouton* avec *AttenteRelacheBouton* positionnée à "False". Dans quelques cas exceptionnels, vous aurez besoin de positionner *AttenteRelacheBouton* à "False" (REF fait partie de ces exceptions).

Fenêtres de souris et coordonnées de souris

Dans le même esprit que TPCRT, TPSOURIS vous permet d'utiliser des fenêtres de souris (partie de l'écran délimitant le champ de liberté de la souris) et de lire la position relative du pointeur de la souris dans la fenêtre actuelle.

Pour mettre en oeuvre une fenêtre de souris, vous appelerez l'une des deux routines : *FenSouris* ou *FenSourisPleinEcran*. *FenSouris* ressemble à *Window*, elle comporte quatre paramètres : *XBas, YBas, XHaut, YHaut*.

Ainsi pour créer un fenêtre (80 x 25), vous devrez utiliser "*FenSouris (1, 1, 80, 25);*". *FenSourisPleinEcran* est une routine "enveloppe" : tout ce qu'elle fait est d'appeler *FenSouris* avec les dimensions actuelles de l'écran comme paramètres. Puisque TPSOURIS appelle cette procédure à votre place lorsque votre programme démarre, vous n'aurez, en général, pas besoin d'appeler cette routine à moins que vous ne vouliez créer une fenêtre plus petite ou que vous vouliez modifier les modes vidéo ou passer d'une fonte 8x8 à une autre et réciproquement.

Comme nous l'avons évoqué dans la section "Lecture des boutons de la souris", vous pouvez généralement définir l'emplacement du pointeur de la souris au moment où le bouton de la souris a été pressé en contrôlant les variables globales *MotToucheSourisX* et *MotToucheSourisY*. Si vous avez besoin de connaître cet emplacement à un autre moment, vous avez plusieurs solutions. La première est d'utiliser *SourisX* et *SourisY* qui fonctionnent comme *WhereX* et *WhereY*. Une deuxième solution, et quelquefois la meilleure, est d'utiliser *SourisXY* qui renvoie à la fois les coordonnées X et Y du pointeur et le statut des boutons de la souris.

Dans chaque cas, il est supposé que vous désirez connaître la position du pointeur de la souris relativement à la fenêtre souris actuelle. Si ce n'est pas le cas, vous pouvez facilement calculer la position en ajoutant *SourisXBas* à la coordonnée X (colonne) et *SourisYBas* à la coordonnées Y (ligne). Ainsi qu'avec *SourisXHaut* et *SourisYHaut*, ces variables contiennent les dimensions actuelles de la fenêtre de souris (*SourisXBas* et *SourisYBas* sont basée sur la valeur 0, ce qui signifie que le coin supérieur gauche de l'écran a pour coordonnées 0,0 ; *SourisXHaut* et *SourisYHaut* sont basées sur la valeur 1).

"Crottes de souris"

Et maintenant de mauvaises nouvelles. La programmation de la souris peut devenir très fastidieuse à la longue, car les souris sont des animaux difficiles à maîtriser, tant que vous n'utilisez pas le pointeur software.

Voici un scénario typique à méditer :

Le pointeur de la souris est supposé en ligne 1, colonne 1. Juste avant cela, le pilote de la souris sauvegarde le couple caractère-attribut de cet emplacement et le restaure avant de repositionner le pointeur. Tant que l'écran n'est pas modifié, le fonctionnement est correct ; mais que se passe-t-il si notre programme écrit quelque chose en ligne 1, colonne 1 alors que la souris occupe cette position ? Le pointeur sera recouvert, naturellement. Cela n'est pas trop grave, puisque le pointeur sera réactivé lors du prochain déplacement. Ce qui est *plus grave*, c'est que le pilote de la souris restaurera le caractère et l'attribut qui *étaient utilisés* en 1,1, ce qui entraînera un

mauvais positionnement du caractère et des couleurs bizarres sur l'écran. Cette perturbation est appelée "crottes de souris".

Mais que faire ? Il ne vous reste plus qu'à enlever le pointeur de la souris chaque fois que cela sera nécessaire. Si l'écran doit être effacé, déroulé, colorié, écrit, sauvegardé ou restauré, vous devrez au préalable appeler *CacheSouris* et ensuite *AffSouris*. C'est pourquoi nous avons précédemment insisté sur l'intérêt d'utiliser un pointeur de souris hardware lorsque cela est possible.

Bien que TPSOURIS soit impuissante pour lutter contre les "crottes de souris", nous pouvons au moins vous donner une recette qui vous aidera dans certains cas. Si vous exécutez des entrées/sorties écran avec une routine telle que *EcritVite*, vous pouvez utiliser ce modèle de programmation:

```
procedure EcrireSouris(S : String; Ligne, Col, Attr : Byte);
var
   SauveCurseurSouris : Boolean;
begin
   SauveCurseurSouris := SourisInstallee and CursSourisOn;
   if SauveCurseurSouris then
      CacheSouris;
   EcritVite(S, Ligne, Col, Attr);
   if SauveCurseurSouris then
      AffSouris;
end;
```

Grâce à cette routine, vous pourrez réduire notablement le nombre de modifications à apporter dans votre programme. En une seule opération de recherche et remplacement sous votre éditeur, vous pourrez prendre en compte tous les ordres *EcritVite* de l'unité toute entière. Le prix à payer sera la rapidité : les écritures écran prendront plus de temps, ainsi que l'activation et la désactivation du pointeur de souris. Au cas où vous auriez par exemple 6 appels successifs à *EcritVite*, vous pourrez désactiver le pointeur de la souris avant le premier appel et le réactiver après le dernier appel.

Gestionnaire d'événements de la souris

Bien que les fonctionnalités de haut niveau décrites ci-dessus soient adaptées à la plupart des applications, vous aurez besoin de connaître la réaction de la souris lorsqu'un événement se produit (un événement se produit dans l'un des trois cas suivants : la souris est déplacée, un bouton de la souris est pressé ou un bouton de la souris est libéré). Pour obtenir ces informations, vous devrez écrire ce que nous appelons un gestionnaire d'événements de souris.

Cela est compliqué, mais TPSOURIS le fait très facilement. Tout ce que vous avez à faire est d'écrire une procédure globale déclarée FAR sans

paramètre et d'appeler *InitEvenementsSouris* pour indiquer à TPSOURIS que vous voulez que le gestionnaire d'événements soit appelé lorsqu'un événement lié à la souris se produit :

```
{$F+}
procedure MonGestionnaireEvenements;
begin
   {Gestionnaire d'évènements}
end;
   ...
InitEvenementsSouris(EvenementsSouris, @MonGestionnaireEvenements);
```

TPSOURIS installera alors son propre gestionnaire d'événements de bas niveau qui traite les données brutes en provenance de la souris, les convertit en une forme utilisable, range les résultats dans des variables globales et appelle votre routine.

InitEvenementsSouris comporte deux paramètres. Le deuxième, *RoutineUtl*, est l'adresse de votre gestionnaire d'événements, le premier, *MasqueEvenement* est un octet qui définit les événements qui vous intéressent. Vous pouvez initialiser cet octet en utilisant une ou plusieurs des constantes suivantes :

```
DesactiveGestionEvenement = $00;
SourisDeplacee = $01;
BoutonGauchePresse = $02;
BoutonGaucheLibere = $04;
BoutonDroitPresse = $08;
BoutonDroitLibere = $10;
BoutonCentralPresse = $20;  {pour souris à trois boutons seulement}
BoutonCentralLibere = $40;  {pour souris à trois boutons seulement}
EvenementsSouris = $7F;
```

Si vous n'êtes intéressé que par le bouton gauche de la souris, vous spécifierez "*BoutonGauchePresse* + *BoutonGaucheLibere*" comme masque d'événements. Si vous êtes intéressé par tous les événements, vous utiliserez *EvenementsSouris* et si vous désirez désactiver votre gestionnaire d'événements à un certain moment, vous utiliserez "*DesactiveGestionEvenement*" (ou passerez nil comme argument *RoutineUlt*).

Une fois que votre gestionnaire d'événement est installé, vous pouvez obtenir toutes les informations dont vous avez besoin sur l'état de la souris en contrôlant les constantes globales typées suivantes :

```
EvenementSouris : TypeEvenementSouris = DesactiveGestionEvenement;
StatutSouris : StatutBouton = SansBouton;
DernXSouris : Byte = 1;
DernYSouris : Byte = 1;
```

EvenementSouris est un octet du même type que *MasqueEvenement*. Vous pouvez le contrôler pour rechercher un événement donné, un déplacement de souris par exemple en effectuant un ET logique entre *EvenementSouris* et la constante appropriée et contrôler que le résultat est différent de 0 :

```
SourisEstDeplacee := EvenementSouris and SourisDeplacee <> 0;
```

StatutSouris vous indique éventuellement quel bouton a été pressé. Il est déclaré comme étant du type énuméré *StatutBouton* :

```
Type
  StatutBouton = (
    SansBouton, BoutonGauche, BoutonDroit, Les2Boutons,
    {Les valeurs qui suivent concernent les souris à 3 boutons}
    BoutonCentral, BoutonsCentralEtGauche, BoutonsCentralEtDroit,
    Les3Boutons};
```

Vous pouvez ensuite déterminer l'emplacement relatif du pointeur de la souris dans la fenêtre en consultant *DernXSouris* et *DernYSouris*.

Quand vous écrirez votre gestionnaire d'événements, n'oubliez pas que les événements peuvent apparaître plusieurs fois par seconde. Ces programmes devront être brefs, ne pas utiliser trop de place sur la pile et ne devront certainement pas appeler des procédures ou des fonctions qui ne sont pas réentrantes : ce qui inclut la plupart des fonctions DOS, une partie de la bibliothèque de *Turbo Pascal* et toute routine qui pourrait être interrompue si la valeur d'une variable globale était modifiée de façon inattendue au milieu de la routine.

Utilisation de la souris dans un programme résident (TSR)

L'écriture d'un programme résident qui utilise la souris n'est pas trop difficile si vous gardez en mémoire certains points clés. Nous les évoquerons dans un moment mais nous supposerons que vous avez déjà une certaine connaissance sur la façon d'écrire des programmes TSR grâce à l'unité TPTSR. Si ce n'est pas le cas, nous vous recommandons la lecture de la documentation relative à cette unité. Mieux encore, ne poursuivez qu'après avoir écrit au moins un programme résident, dans lequel vous êtes prêt à ajouter le support de la souris.

Le premier point à garder en mémoire est que votre programme résident doit être construit de telle façon qu'il s'exécute au-dessus d'un programme qui utilise déjà la souris. Pour éviter de perturber ce programme, vous devrez sauvegarder l'état de la souris et le restaurer à la sortie. Les routines nécessaires sont *TailleBuffSouris*, *SauveEtatSouris* et *RestaureEtatSouris*.

TailleBuffSouris est une fonction qui renvoie le taille du buffer nécessaire pour sauvegarder toutes les données de travail du pilote de la souris. Nous vous recommandons vivement d'appeler ce programme en tête de votre programme et d'allouer le buffer avant d'activer votre programme résident

pour des raisons que vous trouverez évidentes dans quelque temps. *SauveEtatSouris* et *RestaureEtatSouris* font exactement ce que suggèrent leurs noms.

La seconde chose à garder en mémoire est que vous ne pouvez pas mettre en oeuvre la souris comme vous le voulez avant de lancer le programme résident et vous attendre à ce qu'elle soit configurée comme vous le voulez lorsque vous activez le programme. Le programme que vous activez devrait avoir choisi un type de curseur différent ou initialisé une petite fenêtre de souris par exemple. Aussi, vous devrez réinitialiser la souris chaque fois que vous activerez le programme résident.

Finalement, si vous devez utiliser *LitToucheOuBouton*, *SourisPressee* ou *MotToucheSouris*, vous devrez appeler *ActiveAttenteEvenement* en entrée de votre programme résident et *DesactiveAttenteEvenement* à la sortie. Le premier appel sert à utiliser les trois routines d'entrées/sorties, le second est nécessaire pour garantir que *ActiveAttenteEvenement* fera réellement quelque chose lorsque que vous ré-activerez le programme résident. Maintenant, rassemblons ces éléments. Le programme qui suit est un exemple de squelette d'un programme résident qui utilise TPSOURIS :

```
program MonTSR;
uses
   Dos, TpCrt, TpSouris, TpInt, TpTsr;
var
   PES : PtrEtatSouris;
   TaillePES : Word;

{$F+}
procedure PointEntreePopup(var Regs : Registers);
begin
   ReinitVideo;                    {Réinitialisation de CRT}
   {** Sauvegarde de l'écran et du curseur **}
   if SourisInstallee then begin
      SauveEtatSouris(PES, False);
         {Sauve l'état du gestionnaire de la souris}
      InitSouris;                  {Initialisation de la souris}
      ActiveAttenteEvenement;
         {Installe le gestionnaire d'évènements}
      {** initialise une fenêtre pour la souris **}
      {** choisit un curseur pour la souris et le rend visible **}
   end;
```

```
      {** le corps de la popup se place ici **}
      {** Restaure l'écran et le curseur **}
      if SourisInstallee then begin
         DesactiveAttenteEvenement;
            {Désactive le gestionnaire d'évènements}
         RestaureEtatSouris(PES, False);
            {Restaure l'état du gestionnaire de souris}
      end;
   end;
   {$F-}

   begin
      ...
      if SourisInstallee then begin
         TaillePES := TailleBuffSouris;
            {Alloue le Buffer d'état de la souris}
         if TaillePES = 0 then        {Vérification de la taille}
            SourisInstallee := False
         else
         GetMem(PES, TaillePES);      {Allocation du Buffer}
      end;
      ...
   end.
```

Vérifiez maintenant :

- que vous avez bien appelé *ReinitVideo* avant de sauvegarder l'écran
- que vous avez sauvegardé l'état du pilote de la souris après avoir sauvegardé l'écran
- que vous avez restauré l'état de la souris après avoir restauré l'écran

Si vous ne respectez pas scrupuleusement cet ordre, vous risquez de détruire l'écran du programme au-dessus duquel va s'exécuter votre programme résident. Si vous désirez consulter un programme plus complet que MONTSR, étudiez le programme de démonstration REF.

Différences entre les pilotes de souris

En dépit de ce que nous venons de dire dans le dernier paragraphe, il y a d'autres choses très importantes que vous devez savoir à propos des routines *TailleBuffSouris*, *SauveEtatSouris* et *RestaureEtatSouris*. Les appels de fonctions qu'elles utilisent ne sont pas implémentés par tous les pilotes de souris. Ces fonctions n'existaient pas dans les versions précédentes du pilote de souris *MicroSoft*, mais elles sont maintenant nécessaires avec l'écriture de *CodeView* qui est capable de débugger des programmes qui utilisent la souris. Elles sont probablement implémentées dans les versions actuelles de tous les revendeurs de pilotes de souris mais naturellement personne n'a les versions actuelles. En vérité, vous devez être très prudent, lorsque vous utilisez ces fonctions (*LitPageSouris* et *InitPageSouris* appartiennent à cette catégorie) dans vos programmes. Nous nous sommes efforcés de faire en

sorte que *TailleBuffSouris* renvoie 0 lorsqu'elle rencontre un pilote de souris qui ne les implémente pas. Vous devrez offrir à vos utilisateurs le moyen de désactiver manuellement la souris, ou les informer par des messages très précis que vos programmes risquent de ne pas fonctionner correctement avec des pilotes de souris antérieurs.

Le Mickey

Le dernier élément technique. La plus petite unité de distance reconnue par une souris est un "Mickey", environ 1/100 de pouce (1 pouce = 2,54 cm). Quand vous déplacez une souris sur votre bureau, le déplacement physique doit être communiqué au pilote de la souris qui convertira ce déplacement en un certain nombre de Mickeys. Le pilote de la souris doit alors calculer le déplacement réel sur l'écran du pointeur de la souris. Ce calcul fait appel à une valeur appelée "ratio Mickey-en-pixels ".

Le ratio Mickey-en-pixels indique le nombre de Mickeys dont la souris doit être déplacée afin de déplacer le pointeur de la souris de 8 pixels. En vérité, il y a deux ratios : un tenant compte du déplacement horizontal, et un autre du déplacement vertical. Les ratios par défaut sont 8 et 16 respectivement. Dans les mode texte 80 colonnes, cette valeur exige que la souris soit déplacée d'environ 8/100 de pouce horizontalement afin de déplacer le pointeur de la souris d'une colonne de gauche à droite et 16/100 de pouce vers le haut ou vers le bas afin de déplacer d'une colonne.

Vous êtes perdu ? Heureusement, vous n'avez pas besoin de connaître les mécanismes des souris pour les implémenter dans vos programmes. Cette partie sert principalement de support aux routines *InitRatioMickeyEnPixel* et *LitTotalMickey* décrites plus loin et que personne n'aura à utiliser (voir le programme de démonstration CREEMENU dans lequel l'utilisation de la routine *InitRatioMickeyEnPixel* est utile).

TPSOURIS : Déclarations

Types

```
StatutBouton = SansBouton, BoutonGauche, BoutonDroit, Les2Boutons,
               BoutonCentral, BoutonCentralEtGauche,
               BoutonCentralEtDroit, Les3Boutons);
```

Utilisé dans les routines qui veulent vérifier si un bouton de souris donné (ou une combinaison de boutons) a été pressé. Pour une souris à deux boutons, seules les valeurs *SansBouton..Les2Boutons* sont significatives.

```
TypeEvenementSouris = DesactiveGestionEvenement..EvenementsSouris;
```

Un événement souris (déplacement de souris, bouton pressé, bouton libéré).

```
EtatSouris =
   Record
      TailleBuff : Word;
      Buffer : array[1..400] of byte;
   end;
PrtEtatSouris = ^EtatSouris;
```

Structure de données utilisée pour enregistrer l'état du pilote de la souris. Le buffer est un tableau de taille variable. *TailleBuff* est égale à la taille du tableau + 2.

```
SourisXBas = Byte;
SourisXHaut = Byte;
SourisYBas = Byte;
SourisYHaut = Byte;
```

Coordonnées écran de la fenêtre de souris actuelle. Comme avec les variables *WindMin* et *WindMax* de CRT et TPCRT, *SourisXBas* et *SourisYBas* sont basées sur la valeur 0 alors que *SourisXHaut* et *SourisYHaut* sont basées sur la valeur 1.

Constantes

```
MasqueEcranParDefaut = $FFFF;
MasqueCurseurParDefaut = $7700;
```

Valeurs par défaut des masques d'écrans et de curseur pour un pointeur souris software.

```
DesactiveGestionEvenement   = $00;
SourisDeplacee              = $01;
BoutonGauchePresse          = $02;
BoutonGaucheLibere          = $04;
BoutonDroitPresse           = $08;
BoutonDroitLibere           = $10;
```

```
BoutonCentralPresse          = $20;
BoutonCentralLibere          = $40;
EvenementsSouris             = $7F;
```

Masque des différents événements souris.

```
EvenementSouris : TypeEvenementSouris = DesactiveGestionEvenement;
StatutSouris : StatutBouton = SansBouton;
DernXSouris : Byte = 1;
DernYSouris : Byte = 1;
```

Utilisé pour passer des données aux gestionnaires d'événements qui ont été initialisés avec *InitEvenementsSouris*.

```
SourisGauche            = $EF00;    {Bouton gauche}
SourisDroit             = $EE00;    {Bouton droit}
SourisLes2              = $ED00;    {Les deux boutons}
SourisCentral           = $EC00;    {Bouton central}
SourisGaucheCentral     = $EB00;    {Boutons gauche et central}
SourisDroitCentral      = $EA00;    {Boutons droit et central}
SourisLes3              = $E900;    {Les trois boutons}
```

Pseudo-codes de balayage renvoyés par *MotToucheSouris* et *LitToucheOuBouton* lorsqu'un bouton de souris (ou une combinaison de boutons) a été pressé.

```
RoutineSouris : Pointer = nil;
```

Si cette constante vaut nil, *RoutineSouris* pointe sur un gestionnaire d'événements utilisateur initialisé avec *InitEvenenementsSouris*.

```
RoutineEvenementSouris:TypeEvenementSouris = DesactiveGestionEvenement
```

Représentation binaire de l'octet indiquant le ou les événements qui entraînent l'appel de la procédure pointée par *RoutineSouris*.

```
AttenteRelacheBouton : Boolean = True;
```

Si cette constante vaut "True", *MotToucheSouris* attend que le bouton pressé soit relâché avant de renvoyer le pseudo-code de balayage.

Variables

```
CursSourisOn : Boolean;
```

Cette variable indique si le pointeur de la souris est actuellement affiché ou non.

```
SourisInstallee : Boolean;
```

Indique si une souris est installée ou non.

```
MotToucheSourisX : Byte;
MotToucheSourisY : Byte;
```

Contiennent les coordonnées du pointeur de la souris au moment du dernier appel à *MotToucheSouris*.

ActiveAttenteEvenement

Déclaration

```
procedure ActiveAttenteEvenement;
```

Objet

Active le gestionnaire d'événements nécessaire pour les routines *SourisPressee* et *MotToucheSouris*.

Commentaires

Cette routine doit être appelée avant d'utiliser *SourisPressee* et *MotToucheSouris* dans un programme. Elle active le gestionnaire d'événements de la souris de TPSOURIS qui sera appelé par le pilote de la souris chaque fois le pointeur de la souris est déplacé ou qu'un bouton est pressé ou relâché.

Pour plus de détails, voir les explications sur "Lecture des boutons de la souris" et "Gestionnaire d'événements de la souris" au début de la présentation de TPSOURIS.

Exemple

```
ActiveAttenteEvenement;
...
if SourisPressee then begin
   PseudoTouche := MotToucheSouris;
   {Gère la commande affectée à PseudoTouche}
end;
```

Active le gestionnaire d'événements puis utilise *SourisPressee* et *MotToucheSouris* pour obtenir des données sur le bouton de la souris.

Voir aussi

SourisPressee (TPSOURIS)
MotToucheSouris (TPSOURIS)
DesactiveAttenteEvenement (TPSOURIS)
InitEvenementsSouris (TPSOURIS)

AffSouris

Déclaration

```
procedure AffSouris;
```

Objet

Affiche le pointeur de la souris.

Commentaires

Cette routine doit être appelée pour réactiver le pointeur de la souris, que ce soit un curseur software ou un curseur hardware. Pour plus de détails, voir les explications "Le pointeur de la souris" au début de l'unité TPSOURIS.

Exemple

```
CursSourisBloc;
AffSouris;
...
CacheSouris;
```

Sélectionne un pointeur de souris hardware en forme de bloc, visualise le pointeur, fait quelque chose et cache à nouveau le pointeur.

Voir aussi

CacheSouris (TPSOURIS) InitSouris (TPSOURIS)

BoutonSourisPresse

Déclaration

```
function BoutonSourisPresse(Bouton : StatutBouton; var Compte : word;
                            var DernX, DernY : byte): Boolean;
```

Objet

Renvoie "True" si le bouton à contrôler a été pressé.

Commentaires

Cette routine vous permet de savoir si un bouton donné de la souris a été pressé. Si oui, *Compte* renvoie le nombre de fois qu'il a été pressé depuis le dernier appel à *BoutonSourisPresse*. *DernX* et *DernY* contiendront la position du pointeur de la souris la dernière fois qu'il a été pressé.

En général, nous vous recommandons d'utiliser un gestionnaire d'événements si vous êtes tenté d'utiliser cette fonction et la fonction conjointe *BoutonSourisRelache*. Ces deux routines sont complémentaires.

Exemple

```
if BoutonSourisPresse(BoutonGauche, Nombre, DernierX, DernierY) then
   WriteLn('Le bouton gauche de la souris a été pressé ', Nombre,'
           fois.');
```

Affiche le nombre de fois où le bouton gauche de la souris a été pressé.

Voir aussi

SourisXY (TPSOURIS)

BoutonSourisRelache

Déclaration

```
function BoutonSourisRelache(Bouton : StatutBouton; var Compte : Word;
                             var DernX, DernY : Byte): Boolean;
```

Objet

Renvoie "True" si le bouton à contrôler a été relâché.

Commentaires

Cette routine vous permet de savoir si un bouton donné de la souris a été relâché. Si oui, *Compte* renvoie le nombre de fois qu'il a été relâché depuis le dernier appel à *BoutonSourisRelache*. *DernX* et *DernY* contiendront la position du pointeur de la souris la dernière fois qu'il a été relâché.

Exemple

```
if BoutonSourisRelache(BoutonGauche, Nombre, DernierX, DernierY) then
   WriteLn('Le bouton gauche de la souris a été relâché ', Nombre,'
           fois.');
```

Affiche le nombre de fois où le bouton gauche de la souris a été relâché.

CacheSouris

Déclaration

```
procedure CacheSouris;
```

Objet

Cache le pointeur de la souris.

Commentaires

Cette routine cache le pointeur de la souris quel que soit son type ou sa forme. Elle sera le plus souvent utilisée avec un pointeur software lorsque l'écran doit être déroulé ou lorsqu'une copie de l'écran doit être faite (par une routine de TPFENETR par exemple). Pour plus de détails, voir les explications sur "le pointeur de la souris" au début de la présentation de TPSOURIS.

Exemple

```
CacheSouris;
{Ecriture à l'écran, déroulement de l'affichage,
 sauvegarde d'une image, etc}
AffSouris;
```

Cache le pointeur de la souris, fait quelque chose sur l'écran, puis réactive le pointeur de la souris.

Voir aussi

InitSouris (TPSOURIS)

AffSouris (TPSOURIS)

CursSourisBloc

Déclaration

```
procedure CursSourisBloc;
```

Objet

Initialise des lignes de balayage pour former un pointeur de souris en forme de bloc.

Commentaires

Cette routine sélectionne un pointeur de souris hardware en forme de bloc. Elle est semblable à la routine *CursBloc* de TPCRT.

Voir aussi

CursBloc (TPCRT)

CursSourisHard (TPSOURIS)

CursSourisCache

Déclaration

```
procedure CursSourisCache;
```

Objet

Cache le pointeur de la souris.

Commentaires

Cette routine sélectionne un pointeur souris hardware qui est (au moins temporairement) caché. Elle est identique à la routine *CursCache* de TPCRT. Remarquez que cette routine n'a pas le même rôle que la fonction *CacheSouris* qui cache soit un pointeur de souris software soit un pointeur hardware.

Voir aussi

CursSourisHard (TPSOURIS)

CursCache (TPCRT)

CursSourisHard

Déclaration

```
procedure CursSourisHard(LigneDepart, LigneFin : Word);
```

Objet

Initialise la souris pour utiliser un pointeur hardware.

Commentaires

Cette routine fait deux choses : elle sélectionne le curseur souris hardware plutôt que le curseur software (option par défaut du pilote de la souris) et vous permet de spécifier la ligne de départ (*LigneDepart*) et la ligne de fin (*LigneFin*) comme lignes de balayage de ce pointeur. C'est une routine de bas niveau et dans la plupart des cas, vous préfèrerez probablement utiliser une routine de plus haut niveau qui l'appelle : *CursSourisBloc*, *CursSourisPlat*, *CursSourisCache* ou *CursSourisNormal*.

Pour plus de détails, voir les explications sur le pointeur de la souris au début de la présentation de TPSOURIS.

Exemple

```
if Fonte8x8Sel or (ModeActuel <> 7) then
   CursSourisHard(0, 7)
else
   CursSourisHard(0, 12);
```

Comme la routine *CursSourisBloc*, ce programme sélectionnera un pointeur hardware qui aura la forme d'un bloc.

Voir aussi

CursSourisBloc (TPSOURIS)
CursSourisPlat (TPSOURIS)
CursSourisCache (TPSOURIS)
CursSourisNormal (TPSOURIS)
CursSourisSoft (TPSOURIS)

CursSourisNormal

Déclaration

```
procedure CursSourisNormal;
```

Objet

Initialise normalement les lignes de balayage pour le pointeur de la souris en fonction du mode vidéo actuel.

Commentaires

Cette routine sélectionne un pointeur de souris hardware qui a deux lignes de haut. Elle est semblable à la routine *CursNormal* de TPCRT.

Voir aussi

CursSourisHard (TPSOURIS)

CursSourisPlat

Déclaration

```
procedure CursSourisPlat;
```

Objet

Initialise les lignes de balayage pour former un curseur plat en fonction du mode vidéo courant.

Commentaires

Cette routine sélectionne un pointeur de souris hardware qui est légèrement plus large que le curseur normal. Elle est semblable à la routine *CursPlat* de TPCRT.

Voir aussi

CursPlat (TPCRT)

CursSourisHard (TPSOURIS)

CursSourisSoft

Déclaration

```
procedure CursSourisSoft(MasqueEcran, MasqueCurseur : Word);
```

Objet

Initialise la souris pour utiliser un pointeur software.

Commentaires

Cette routine sélectionne un pointeur software (par défaut) plutôt qu'un curseur hardware et vous permet de spécifier la forme et la couleur du pointeur.

Pour plus de détails sur les pointeurs de souris, y compris les masques d'écran et les masques de curseur, voir les explications sur "Le pointeur de la souris" au début de l'unité TPSOURIS.

Exemple

```
CursSourisSoft(MasqueEcranParDefaut, MasqueCurseurParDefaut);
```

Sélectionne le type de curseur software par défaut, un bloc non clignotant dont les couleurs varient en fonction de la position du pointeur.

```
CursSourisSoft($00FF, $F000);
```

Sélectionne un curseur bloc clignotant en inversion vidéo (blanc/noir).

```
CursSourisSoft($0000, $1F04);
```

Sélectionne un curseur en forme de losange blanc (ASCII4) sur un fond bleu.

Voir aussi

CurseurSourisHard (TPSOURIS)

DesactiveAttenteEvenement

Déclaration

```
procedure DesactiveAttenteEvenement;
```

Objet

Désactive le gestionnaire d'événements installé par *ActiveAttenteEvenement*.

Commentaires

Cette routine désactive le gestionnaire d'événements interne de TPSOURIS qui est installé lorsque *ActiveAttenteEvenement* est appelée. Elle sert principalement dans les programmes résidents qui doivent utiliser *SourisPressee* et *MotToucheSouris*. Voir *ActiveAttenteEvenement* comme exemple.

Voir aussi

ActiveAttenteEvenement (TPSOURIS)
InitEvenementsSouris (TPSOURIS)

FenSouris

Déclaration

```
procedure FenSouris(XBas, YBas, XHaut, YHaut : Byte);
```

Objet

Initialise les coordonnées de la fenêtre dans laquelle la souris devra se déplacer.

Commentaires

Cette routine est semblable à la procédure *Window* de CRT et TPCRT. Elle positionne les valeurs horizontales et verticales, minimum et maximum de l'écran pour le pointeur de la souris. Les coordonnées d'écran doivent suivre les mêmes règles que celles passées à *Window* : *XBas/YBas* sont inférieures ou égales à *XHaut/YHaut*, *XHaut* est inférieure ou égale à *LargeurEcran* et *YHaut* est inférieure ou égale à *HauteurEcran*. Si l'une de ces coordonnées est incorrecte, *FenSouris* est inopérante. Si les coordonnées sont valides, le pointeur de la souris sera déplacé dans la fenêtre s'il n'y est pas déjà et les coordonnées de la nouvelle fenêtre seront enregistrées dans *SourisXBas* et *SourisXHaut*, *SourisYBas* et *SourisYHaut*.

La fenêtre souris par défaut est l'écran tout entier, basée sur la largeur et la hauteur de l'écran lorsque le programme commence. Si les dimensions de l'écran changent ultérieurement, vous devrez appeler *FenSourisPleinEcran* pour initialiser les coordonnées de la fenêtre souris.

Exemple

```
Window(1, 1, 80, 25);
if SourisInstallee then
   FenSouris(1, 1, 80, 25);
```

Crée une fenêtre 80x25 et une fenêtre souris correspondante. Remarquez que *FenSouris* n'est appelée que si *SourisInstallee* vaut "True".

Voir aussi

FenSourisPleinEcran (TPSOURIS)

FenSourisPleinEcran

Déclaration

```
procedure FenSourisPleinEcran;
```

Objet

Sélection de tout l'écran comme fenêtre de souris.

Commentaires

Cette routine sélectionne l'écran comme fenêtre de souris courante. Remarquez qu'il n'est pas nécessaire d'appeler explicitement cette routine en début de programme puisqu'elle a déjà été appelée si la souris est installée.

Exemple

```
SelFonte8x8(True);
FenSourisPleinEcran;
```

Passe en mode 43/50 lignes et réinitialise les coordonnées de la fenêtre de souris.

Voir aussi

FenSouris (TPSOURIS)

InitEvenementsSouris

Déclaration

```
procedure InitEvenementsSouris(MasqueEvenement : TypeEvenementSouris;
      RoutineUtl : Pointer);
```

Objet

Initialise l'adresse de la routine à appeler lorsqu'un événement lié à la souris se produit.

Commentaires

Contrairement à la routine *ActiveAttenteEvenement*, l'objet principal de cette routine n'est pas d'activer notre propre gestionnaire d'événements de TPSOURIS (bien qu'il le sera si nécessaire), mais plutôt de vous permettre de spécifier votre propre routine (pointée par *RoutineUtl*) qui sera appelée lorsqu'un événement de la souris se produira.

Si *RoutineUtl* a une valeur différente de nil, elle devra pointer sur une procédure déclarée FAR sans paramètre. Les données sur l'événement qui est apparu, peuvent être obtenues en contrôlant les variable globales *EvenementSouris*, *StatutSouris*, *DernXSouris* et *DernYSouris*. Si *RoutineUtl* vaut nil (ou *MasqueEvenement* = *DesactiveGestionEvenement*), tout gestionnaire d'événements défini par l'utilisateur actif sera désactivé (il ne sera pas appelé).

MasqueEvenement est un octet binaire qui indique au gestionnaire d'événements de TPSOURIS quand votre routine doit être appelée. Par exemple, si vous voulez qu'elle soit appelée seulement lorsque le bouton gauche est pressé ou libéré, *MasqueEvenement* devrait avoir la valeur "*BoutonGauchePresse* + *BoutonGaucheLibere*". Si vous désirez connaître tous les événements de la souris, vous utiliserez à la place *EvenementsSouris*.

Remarquez que TPSOURIS vous décharge de l'écriture d'un gestionnaire d'événements : elle sauvegarde tous les registres CPU, initialise le registre DS pour pointer sur votre propre segment de données, appelle *RoutineUtl*, puis restaure les registres CPU lorsqu'elle reprend le contrôle.

Pour plus de détails, voir les explications de "Gestionnaire d'événements de souris" au début de l'unité TPSOURIS.

Voir aussi

DesactiveAttenteEvenement (TPSOURIS)
ActiveAttenteEvenement (TPSOURIS)

InitPageSouris

Déclaration

```
procedure InitPageSouris(Page : Byte);
```

Objet

Initialise le page vidéo où la souris sera affichée.

Commentaires

Cette routine, semblable à la routine *InitPageVisible* de TPCRT, indique au pilote de la souris quelle page vidéo utiliser lors de l'affichage du pointeur de la souris.

Avertissement : Cette routine utilise une fonction qui n'est pas implémentée dans tous les pilotes de souris. Voir les explications de *TailleBuffSouris* pour plus de détails.

Exemple

```
InitPageVisible(3);
InitPageSouris(PageActuelle);
```

Bascule à la fois l'affichage et le pointeur de la souris en page vidéo 3. Remarquez que *InitPageSouris* utilise *PageActuelle* au cas où l'appel à *InitPageVisible* ne fonctionnerait pas.

Voir aussi

LitPageSouris (TPSOURIS)

InitPageVisible (TPCRT)

InitRatioMickeyEnPixel

Déclaration

```
procedure InitRatioMickeyEnPixel(Horizontal, Vertical : Integer);
```

Objet

Initialise le ratio Mickey-en-pixels.

Commentaires

Pour des explications sur les "Mickeys" et "ratio Mickey-en-pixels", voir la partie "Mickey" au début de l'unité TPSOURIS.

Cette routine vous permet de modifier les ratios Mickey-en-pixels. Bien que les valeurs par défaut de 8, 16 (horizontal, vertical) soient en général appropriées, vous pourrez dans certains cas les incrémenter ou les décrémenter pour régler les déplacements de la souris. Des valeurs de 16, 32 la feront glisser doucement vers le bas et rendront la souris moins sensible aux déplacements alors que des valeurs de 4, 8 la déplaceront plus rapidement et la rendront plus sensible aux déplacements.

Les paramètres *Horizontal* et *Vertical* doivent être des entiers appartenant à l'intervalle 1 à 32 767 (*MaxInt*).

Exemple

```
InitRatioMickeyEnPixel(4, 8);
```

Provoque l'accélération.

```
InitRatioMickeyEnPixel(4, 8);
```

Provoque le ralentissement.

InitSouris

Déclaration

```
procedure InitSouris;
```

Objet

Reinitialise la souris et positionne *SourisInstallee.*

Commentaires

Cette routine réinitialise la souris et positionne *SourisInstallee.* Bien qu'elle soit appelée automatiquement en début de programme, dans certains cas, vous pourrez l'appeler ultérieurement comme par exemple dans un programme résident.

Cette routine effectue les opérations suivantes :

- Elle cache le pointeur de la souris
- Elle sélectionne un pointeur souris software (encore caché) avec un masque d'écran égal à $FFFF (masque d'écran par défaut) et un masque curseur égal à $7700 (masque de curseur par défaut).
- Elle sélectionne une fenêtre souris de 80x25 (40x25 en mode 40 colonnes)
- Elle déplace le pointeur au centre de cette fenêtre
- Elle sélectionne un ratio Mickey-en-pixels de 8,16
- Elle désactive tout gestionnaire d'événements défini par l'utilisateur

Exemple

```
SauveEtatSouris(PES, False);
InitSouris;
ActiveAttenteEvenement;
CursSourisBloc;
AffSouris;
...
DesactiveAttenteEvenement;
RestaureEtatSouris(PES, False);
```

Suite typique d'instructions dans un programme résident qui utilise la souris : sauvegarde l'état du pilote de la souris, réinitialise la souris puis sélectionne un pointeur de souris hardware et l'affiche. Plus loin, l'état précédent du pilote de la souris est restauré.

Voir aussi

CacheSouris (TPSOURIS)

AffSouris (TPSOURIS)

LitPageSouris

Déclaration

```
function LitPageSouris : Byte;
```

Objet

Renvoie la page vidéo dans laquelle la souris a été affichée.

Commentaires

Cette routine renvoie le numéro de la page vidéo où le pointeur de la souris est actuellement affiché.

Avertissement : Cette routine utilise une fonction souris qui n'est pas implémentée dans tous les pilotes de souris. Voir les explications de *TailleBuffSouris* pour plus de détails.

Exemple

```
SauvePageSouris := LitPageSouris;
InitPageSouris(NvellePageSouris);
...
InitPageSouris(SauvePageSouris);
```

Ceci sauvegarde la page souris courante, passe à une nouvelle page puis, restaure l'ancienne page.

Voir aussi

InitPageSouris (TPSOURIS)

LitTotalMickey

Déclaration

```
procedure LitTotalMickey(var Horizontal, Vertical : Integer);
```

Objet

Renvoie le total de Mickeys horizontaux et verticaux depuis le dernier appel à cette fonction.

Commentaires

Cette routine renvoie le nombre de Mickeys dont la souris a été déplacée depuis le dernier appel à *LitTotalMickey*. Des nombres négatifs indiquent un mouvement vers le haut (horizontal) ou vers la gauche (vertical). Des nombres positifs indiquent un mouvement vers la bas (horizontal) ou vers la droite (vertical).

LitTotalMickey est très utile lorsque vous devez savoir si l'utilisateur a déplacé ou non la souris même si la position du pointeur de la souris reste inchangée, au cas où il aurait atteint le bord de la fenêtre. Par exemple, si le pointeur de la souris se trouve en ligne 1, colonne 1 et que *LitTotalMickey* renvoie une valeur horizontale de 0 et une valeur verticale de -16, vous pouvez en déduire que l'utilisateur a essayé de déplacer la souris de bas en haut.

Voir la section "Le Mickey" dans la présentation de TPSOURIS pour la définition du terme Mickey.

Voir aussi

InitRatioMickeyEnPixel (TPSOURIS) SourisXY (TPSOURIS)

LitToucheOuBouton

Déclaration

```
function LitToucheOuBouton : Word;
```

Objet

Renvoie le code de la touche ou du bouton souris pressé.

Commentaires

Cette routine est fournie partiellement pour des utilisations internes et partiellement pour vos propres besoins. Elle attend en boucle qu'une touche ou qu'un bouton de la souris soit pressé, puis renvoie soit le code de balayage de la touche si une touche a été pressée en premier, soit un pseudo-code de balayage du bouton de la souris si un bouton a été pressé en premier. Si le bouton de la souris a été pressé en premier, à la sortie, la position actuelle du pointeur de la souris sera enregistrée dans *MotToucheSourisX* et *MotToucheSourisY*.

LitToucheOuBouton utilise *SourisPressee* et *MotToucheSouris* pour obtenir les données sur les boutons de la souris et respecte en conséquence les mêmes règles : *ActiveAttenteEvenement* doit être appelée en premier pour qu'elle puisse obtenir les informations sur les boutons de la souris.

Voir les explications sur "Lecture des boutons de la souris" au début de l'unité TPSOURIS pour plus de détails.

Exemple

```
I := Integer(LitToucheOuBouton);
if Lo(I) = 0 then
   case I of
      $0000 : {^Break pressé} ;
      ...
      Integer(SourisGauche) then {Bouton gauche de la souris pressé} ;
   end;
```

Lit la prochaine touche ou le bouton de la souris pressé et exécute la commande. Remarquez que *SourisGauche* doit être transformée en un entier puisque *Turbo Pascal* ne permet pas d'utiliser des étiquettes de type word dans un sélecteur de cas.

Voir aussi

ActiveAttenteEvenement (TPCRT)
LitMotTouche (TPCRT)
MotToucheSouris (TPCRT)

MotToucheSouris

Déclaration

```
function MotToucheSouris : Word;
```

Objet

Renvoie le pseudo-code de balayage correspondant au bouton de la souris qui a été pressé.

Commentaires

Cette routine est semblable à la fonction *LitMotTouche* de TPCRT. Si *ActiveAttenteEvenement* a été appelée, elle attend en boucle qu'un bouton de la souris soit pressé, puis renvoie le pseudo-code de balayage dans l'intervalle *SourisGauche..SourisLes3* qui précise quelle combinaison de boutons a été pressée. Si *ActiveAttenteEvenement* n'a pas été appelée, elle renverra $FFFF.

Une fois qu'un bouton de la souris a été pressé, *MotToucheSouris* exécute l'une des deux actions en fonction de la valeur actuelle de *AttenteRelacheBouton*. Si cette variable vaut "False", elle renverra immédiatement, le pseudo-code de balayage pour la combinaison de boutons qui a été pressée. Si elle vaut "True", *MotToucheSouris* continuera à contrôler le statut des boutons de la souris jusqu'à ce qu'ils soient libérés. Elle rangera à la sortie la position actuelle du pointeur de la souris dans *MotToucheSourisX* et *MotToucheSourisY*.

Voir les explications sur "Lecture des boutons de la souris" au début de l'unité TPSOURIS pour plus de détails.

Exemple

```
CodeBalayage := $FFFF;
repeat
   if KeyPressed then
      CodeBalayage := LitMotTouche
   else if SourisPressee then
      CodeBalayage := MotToucheSouris;
until CodeBalayage <> $FFFF;
```

Boucle jusqu'à ce qu'une touche ou un bouton de la souris soit pressé (c'est essentiellement ce que fait *LitToucheOuBouton*).

Voir aussi

ActiveAttenteEvenement (TPSOURIS)
SourisXY (TPSOURIS)
LitToucheOuBouton (TPSOURIS)

RestaureEtatSouris

Déclaration

```
procedure RestaureEtatSouris(var PES : PtrEtatSouris; Libre: Boolean);
```

Objet

Restaure l'état du pilote de la souris et libère le buffer.

Commentaires

Cette fonction restaure l'état du pilote de la souris. Elle est destinée principalement aux programmes résidents qui doivent utiliser le support de la souris. En entrée, *PES* pointe sur le buffer contenant les données à restaurer. Si *Libre* vaut "True", ce buffer sera libéré à la sortie et *PES* sera reinitialisée à nil. Si la souris n'est pas installée, ou si *PES* vaut nil à l'entrée, *RestaureEtatSouris* est inopérante.

Avertissement : cette routine utilise une fonction souris qui n'est pas implémentée avec tous les pilotes de souris. Voir les explications de *TailleBuffSouris* pour plus de détails.

Pour un exemple d'utilisation de cette routine, voir l'introduction sur "Utilisation de la souris dans les programmes résidents".

SauveEtatSouris

Déclaration

```
procedure SauveEtatSouris(var PES : PtrEtatSouris; Libre : Boolean);
```

Objet

Sauve l'état du pilote de la souris et alloue le buffer.

Commentaires

Cette fonction sauve l'état du pilote de la souris. Elle est destinée principalement aux programmes résidents qui doivent utiliser le support de la souris. *PES* pointe sur le buffer utilisé pour sauvegarder les données du pilote de la souris. Si *Libre* vaut "True", ce buffer sera alloué sur le tas et *PES* pointera sur celui-ci à la sortie. Si *Libre* vaut "False", *PES* devrait pointer sur un buffer alloué précédemment. Si la souris n'est pas installée, ou si *TailleBuffSouris* renvoie 0, *PES* sera initialisée à nil.

Avertissement : Cette routine utilise une fonction souris qui n'est pas implémentée avec tous les pilotes de souris. Voir les explications de *TailleBuffSouris* pour plus de détails.

Comme exemple d'utilisation de cette routine, voir l'introduction de la partie "Utilisation de la souris dans un programme résident".

Voir aussi

TailleBuffSouris (TPSOURIS)

SourisDansFen

Déclaration

```
function SourisDansFen(XBas, YBas, XHaut, YHaut : Byte): Boolean;
```

Objet

Renvoie "True" si le pointeur de la souris est à l'intérieur de la fenêtre spécifiée.

Commentaires

Cette routine renvoie "True" si le pointeur de la souris est actuellement dans la partie spécifiée de l'écran. Remarquez que les coordonnées écran passées en paramètres sont considérées comme des coordonnées relatives par rapport à la fenêtre souris actuelle.

Exemple

```
FenSouris(5, 5, 75, 20);
...
if SourisDansFen(1, 1, 3, 3) then {faire quelque chose} ;
```

Vérifie si le pointeur de la souris se trouve dans les trois premières lignes et dans les trois colonnes de la fenêtre souris actuelle.

Voir aussi

FenSouris (TPSOURIS)

SourisEnXY

Déclaration

```
procedure SourisEnXY(SourisX, SourisY : Byte);
```

Objet

Positionne la souris.

Commentaires

Cette routine est semblable à la routine *GotoXY* de CRT et TPCRT. Elle déplace le pointeur de la souris à la position spécifiée. Comme pour toute l'unité TPSOURIS, les coordonnées écran spécifiées sont des coordonnées relatives à la fenêtre souris actuelle. Si les coordonnées sont en dehors de la fenêtre souris, *SourisEnXY* est inopérante.

Exemple

```
FenSouris(5, 5, 75, 20);
SourisEnXY(1, 1);
```

Place le pointeur de la souris en ligne 5, colonne 5.

Voir aussi

FenSouris (TPSOURIS)

SourisPressee

Déclaration

```
function SourisPressee : boolean;
```

Objet

Renvoie "True" si le bouton de la souris est pressé.

Commentaires

Cette routine est semblable à la fonction *KeyPressed* de CRT et TPCRT. Elle vous permet de savoir si un bouton de la souris est pressé. Comme *MotToucheSouris*, elle est inopérante si *ActiveAttenteEvenement* n'a pas été appelée (dans ce cas, elle renverra toujours "False"). Voir *MotToucheSouris* en exemple.

Voir aussi

ActiveAttenteEvenement (TPSOURIS)
MotToucheSouris (TPSOURIS)

SourisX

Déclaration

```
function SourisX : Byte;
```

Objet

Renvoie la coordonnée X actuelle de la souris.

Commentaires

Cette routine est semblable à la fonction *WhereX* de CRT et TPCRT. Elle renvoie la coordonnée actuelle X (colonne) du pointeur de la souris relative à la fenêtre souris actuelle. Pour obtenir une coordonnée absolue de la souris, ajoutez *SourisXBas* à la valeur renvoyée par *SourisX*.

Voir aussi

SourisY (TPSOURIS)

FenSouris (TPSOURIS)

SourisXY

Déclaration

```
procedure SourisXY(var X, Y : Byte; var Statut : StatutBouton);
```

Objet

Renvoie la position de la souris et le statut des boutons

Commentaires

Cette routine renvoie la position actuelle du pointeur de la souris relative à la fenêtre souris actuelle dans *X* et *Y* (pour obtenir les coordonnées ligne/colonne absolues de la souris, ajoutez *SourisXBas* et *SourisYBas* aux valeurs renvoyées dans *X* et *Y*). L'état courant des boutons de la souris est renvoyé dans *Statut*. Cette routine est particulièrement utile si vous voulez savoir quels boutons ont été pressés et exécuter différentes fonctions par rapport à la position du pointeur de la souris.

Exemple

```
SourisXY(X, Y, Statut);
case Statut of
   BoutonGauche : GererBoutonGauche;       {Bouton gauche pressé}
   BoutonDroit  : GererBoutonDroite;       {Bouton droit pressé}
   Les2Boutons  : GererLesDeuxBoutons;
                                   {Boutons droit et gauche pressés}
end;
```

Donne l'emplacement du pointeur de la souris et le statut des boutons de la souris, puis exécute une procédure donnée en fonction de la combinaison de boutons qui a été pressée.

Voir aussi

SourisY (TPSOURIS)

FenSouris (TPSOURIS)

SourisY

Déclaration

```
function SourisY : Byte;
```

Objet

Renvoie la coordonnée Y actuelle de la souris.

Commentaires

Cette routine est semblable à la fonction *WhereY* de CRT et TPCRT. Elle renvoie la coordonnée actuelle Y (ligne) du pointeur de la souris relative à la fenêtre souris actuelle. Pour obtenir une coordonnée absolue de la souris, ajoutez *SourisYBas* à la valeur renvoyée par .

Voir aussi

SourisX (TPSOURIS)

SourisXY (TPSOURIS)

TailleBuffSouris

Déclaration

```
function TailleBuffSouris : Word;
```

Objet

Renvoie le nombre d'octets de mémoire nécessaires pour sauvegarder l'état du pilote de la souris.

Commentaires

Cette fonction renvoie la taille du buffer nécessaire pour sauvegarder l'état du pilote de la souris. Elle est destinée principalement aux programmes résidents qui doivent utiliser le support de la souris. La valeur retournée variera d'un pilote de souris à un autre mais devra grossièrement se situer dans l'intervalle 350 à 400 octets. Si la souris n'est pas installée, elle renverra 0.

Avertissement : Cette routine utilise une fonction de la souris qui n'est pas implémentée dans tous les pilotes de souris. Elle est écrite de telle façon qu'elle renvoie 0 si la fonction n'est pas implémentée, mais nous ne pouvons garantir ce mode de fonctionnement. Vous devriez en conséquence vérifier si la valeur renvoyée vaut 0 et avertir les utilisateurs des problèmes possibles. Si un problème se pose, ils pourront probablement le corriger simplement en mettant à jour leur logiciel de pilote de souris.

Pour un exemple d'utilisation de cette routine, voir l'introduction de "Utilisation de la souris dans les programmes résidents".

Voir aussi

RestaureEtatSouris (TPSOURIS) SauveEtatSouris (TPSOURIS)

TPPICK : Listes Pick

TPPICK met en oeuvre des routines de bas niveau lorsqu'un choix dans une liste est demandé. Que la liste contienne un répertoire de fichiers, des options de configuration ou des départements et codes postaux, TPPICK offre la souplesse nécessaire pour afficher la liste, permettre à l'utilisateur de la balayer et de faire son choix.

TPPICK et TPMENU sont complémentaires. TPMENU permet de sélectionner un élément dans un menu hiérarchisé mais les listes sont prédéterminées. La liste de TPPICK ne comprend qu'un seul niveau mais peut contenir un nombre arbitraire d'éléments et les éléments peuvent être déterminés dynamiquement juste avant d'afficher et de sélectionner un élément dans la liste Pick. TPPICK est plus simple à utiliser avec des systèmes de menus statiques à un seul niveau.

La routine principale de TPPICK est *FenPick*. Un seul appel à cette routine ouvre une fenêtre, affiche une liste d'éléments, accepte les touches pour "browser" la liste, restaurer l'écran et renvoie l'élément choisi.

TPPICK offre quatre possibilités pour gérer avec souplesse différentes listes. Premièrement, la routine *FenPick* comporte des paramètres qui spécifient la position de la fenêtre, les couleurs et la source de la liste d'éléments. Deuxièmement, votre programme génère la liste d'éléments en fournissant une fonction pour associer un index d'éléments avec une chaîne à afficher. Troisièmement, la partie interface de TPPICK déclare un nombre de constantes typées que votre programme peut modifier pour adapter les différentes possibilités d'affichage de la liste Pick : nombre de colonnes d'éléments, les nombres minimum et maximum de lignes d'écran à utiliser, recherche d'éléments autorisée dans la liste, etc... Comme les autres unités de *Turbo Professionnel*, TPPICK comporte une interface clavier personnalisable. Vous pouvez modifier les touches, ajouter ou supprimer des commandes et modifier la source des entrées clavier en utilisant les méthodes décrites en appendice C. TPPICK supporte aussi la souris et inclut une bande de défilement pour des accès rapides aux autres parties d'une longue liste.

Le programme qui suit est un exemple d'utilisation de TPPICK. L'utilisateur peut faire un choix parmi cinq pizza. Les sélections sont faites en utilisant les touches du curseur ou en pressant la première lettre du nom d'un élément.

```
program ExemplePick;
uses
   TpCrt, TpChaine, TpFenetr, TpCmd, TpPick;
const
   NombreGarniturePizza = 5; {Nombre d'éléments dans la liste pick}
   CouleursPick : TabCouleurPick =
      ($07,                  {Couleur de l'élément non sélectionné}
       $0F,                  {Couleur du cadre de la fenêtre}
       $70,                  {Couleur du titre de la fenêtre}
       $70,                  {Couleur de l'élément sélectionné}
       $07,                  {Deuxième couleur de l'élément non
                              sélectionné}
       $0F                   {Deuxième couleur de l'élément sélectionné}
      );
var
   Choix : Word;             {Elément choisi}

{$F+}
function GarniturePizza(Item : Word) : string;
   {Retourne le nom de chaque élément}
begin
   case Item of
      1 : GarniturePizza := ' Poivrons';
      2 : GarniturePizza := ' Saucisses';
      3 : GarniturePizza := ' Champignons';
      4 : GarniturePizza := ' Anchois';
      5 : GarniturePizza := ' Oignons';
   end;
end;
{$F-}

begin
   ClrScr;
   WriteLn('Choisissez une garniture');

   TpPick.RechPick := CaracPickActuel;
            {Accepte la sélection sur la première lettre du mot}
   Choix := 1;              {Choix initial}

   if not FenPick(@GarniturePizza,         {Retourne les noms des
                                            éléments}
                  NombreGarniturePizza,    {Nombre d'éléments dans la
                                            liste pick}
                  3, 5, 17, 12,            {Coordonnées de la fenêtre}
                  True,                    {Dessine un cadre autour de
                                            la liste}
                  CouleursPick,            {Couleurs à utiliser}
                  '',                      {Pas de titre pour cette
                                            fenêtre}
                  Choix)                   {Elément choisi}
   then
      {Mémoire insuffisante pour le buffer d'écran}
      WriteLn('Impossible d''exécuter FenPick')
```

```
    else
       if NumCmdPick = TPKSelect then
          {Un choix a été fait}
          WriteLn('Garniture choisie : ',GarniturePizza(Choix))
       else
          {Escape pressé}
          WriteLn('Aucun choix');
end.
```

Les routines Pick peuvent utiliser six attributs vidéo séparés pour afficher la liste. La constante *CouleursPick* définit les couleurs. Vous pouvez disposer de deux constantes, l'une utilisée pour un moniteur couleur et l'autre pour un moniteur monochrome.

L'exemple de la fonction *GarniturePizza* est important. Chaque fois que vous utilisez les routines Pick, vous devez définir une fonction qui ressemble à ce modèle : une fonction compilée sous le modèle FAR qui comporte un seul paramètre de type word et renvoie une chaîne. Vous passez l'adresse de votre routine comme paramètre à *FenPick*. Votre routine sera appelée chaque fois que des routines Pick ont besoin de lire le nom d'un élément sur la liste Pick. Le paramètre passé à la routine sera toujours dans l'intervalle 1..*NbElem* où *NbElem* est un autre paramètre que vous passez à *FenPick*. La chaîne que vous renvoyez peut être calculée comme vous voulez : en utilisant une instruction **case** comme dans l'exemple et en recherchant les noms dans un tableau de chaînes de caractères, ou en la calculant en fonction d'autres variables. *FenPick* n'ajoutera pas d'espace en tête de la chaîne renvoyée mais complétera cette chaîne ou éliminera les blancs pour l'ajuster à la longueur voulue.

FenPick comporte plusieurs options de recherche. Par défaut, il n'y a pas de recherche. Aussi, la seule façon de déplacer la barre lumineuse Pick est d'utiliser les touches du curseur. Après avoir sélectionné d'autres options de TPPICK, les routines Pick déplaceront la barre lumineuse lorsque l'utilisateur entrera des caractères alphanumériques qui correspondent aux noms des éléments de menus. Dans l'exemple, nous initialisons la constante *RechPick* déclarée dans l'unité TPPICK à *CaracPickActuel*. Cette option signifie que lorsqu'un caractère alphanumérique correspondant à la première lettre d'un élément Pick est entré, la routine Pick s'arrêtera immédiatement en retournant l'index du prochain élément correspondant. Ce mode de fonctionnement est semblable à celui de TPMENU.

Dans l'exemple, nous indiquons l'élément à mettre en valeur initialement en affectant la valeur 1 à *Choix*.

L'exemple appelle ensuite la routine *FenPick*. Les paramètres de cet appel comportent l'adresse de la fonction *GarniturePizza*, le nombre d'éléments de cette liste, les coordonnées supérieure gauche et inférieure droite de la

fenêtre Pick, le cadre de fenêtre ou non, les valeurs des couleurs utilisées, un titre à afficher sur le bord de la fenêtre et finalement un paramètre **var** qui contient le choix initial et la sélection finale de l'utilisateur.

FenPick renverra "False" en cas d'erreur, parce que les paramètres passés sont incorrects ou parce qu'il n'y a pas assez de place sur le tas pour allouer le buffer d'écran recouvert par la fenêtre Pick.

Si la fenêtre Pick est affichée normalement et que l'utilisateur exécute une action pour quitter la fenêtre, la routine *FenPick* renvoie "True". La variable globale *NumCmdPick* interfacée avec TPPICK contiendra un code indiquant l'action qui a été exécutée pour quitter la fenêtre. Dans cet exemple, nous tenons compte seulement de l'action *TPKSelect* qui signifie que l'utilisateur a pressé (ENTER) ou une touche alphanumérique qui correspond à l'un des éléments. Puis, nous pouvons utiliser le numéro de choix retourné par *FenPick* ainsi que la fonction *GarniturePizza* pour afficher le nom de l'option sélectionnée.

Cet exemple permet simplement de survoler les possibilités de TPPICK. L'unité TPDIR est un exemple plus complet d'utilisation de TPPICK pour sélectionner des noms de fichiers.

TPPICK : Déclarations

Types

```
TypeCouleurPick = (
        AttrFen,          {Couleurs pour les éléments non
                           sélectionnés normalement}
        AttrCadre,        {Couleurs pour le cadre de fenêtre}
        AttrEntete,       {Couleurs pour l'entête de fenêtre}
        AttrSelect,       {Couleurs pour l'élément sélectionné
                           normalement}
        AltNormal,        {Couleurs pour les éléments non
                           sélectionnés }
        AtlHaut,          {Couleurs pour l'élément sélectionné }
TabCouleurPick = Array[TypeCouleurPick] of byte;
```

TypeCouleurPick définit les différentes couleurs utilisées dans une fenêtre Pick. *TabCouleurPick* permet de rassembler ces couleurs en une seule structure. Les couleurs autres que celles de *AttrCadre* et *AttrEntete* demandent certaines explications. Les routines Pick affichent normalement tous les éléments en utilisant l'attribut associé à *AttrFen*, sauf pour l'élément sélectionné actuellement qui est affiché en utilisant la couleur *AttrSelect*. De plus, les routines Pick fournissent un autre jeu de couleurs désigné par les éléments *AltNormal* et *AltHaut*. Chaque élément de la liste Pick peut être choisi en positionnant la constante booléenne *AttrPickAlt* (décrite ci-dessous) à "True" toutes les fois qu'un élément est renvoyé par la fonction utilisateur. De tels éléments sont affichés en utilisant l'attribut *AltNormal* lorsque qu'ils ne sont pas sélectionnés et l'attribut *AltHaut* lorsqu'ils sont sélectionnés.

```
PKType = TPKRien..TPKUtl3;
```

PKTYPE définit la classe des actions exécutées par la routine Pick. Elle comporte des actions telles que curseur en haut et curseur en bas, échappement et sélection d'éléments. Voir les constantes ci-dessous pour plus de détails.

```
ChaineRech = string[LongMaxRech];
```

En combinaison avec la constante *LongMaxRech*, ce type définit la longueur de chaîne maximum utilisée dans une recherche incrémentale.

```
TypeRech = PasRechPick,     {Caractère alphanumérique ignoré}
           ChaineRechPick,  {Recherche incrémentale}
           ChaineRechAlt,   {Recherche incrémentale}
           CaracRechPick,   {Caractère de recherche simple}
           CaracPickActuel); {Caractère de recherche simple,
                              sortie si égalité}
```

TypeRech définit les différents types de recherche que TPPICK peut employer pendant les opérations de recherche. *PasRechPick* signifie que tous les caractères autres que les touches du curseur, (ENTER) et (ESC) sont ignorés.

ChaineRechPick met en oeuvre une recherche incrémentale : lorsque le premier caractère alphanumérique est entré, la barre lumineuse se déplace sur le prochain élément qui commence par ce caractère. Un autre caractère alphanumérique déplace la barre lumineuse sur l'élément suivant qui commence avec les deux caractères saisis et ainsi de suite. Cette approche est pratique pour des listes longues qui ont été triées dans l'ordre alphabétique. Lorsque *ChaineRechPick* est sélectionnée, TPPICK écrira la chaîne de recherche incrémentale courante dans la partie inférieure gauche du cadre de la fenêtre tant qu'il aura assez de place. *ChaineRechAlt* est une autre forme de recherche incrémentale qui combine certains aspects des méthodes de recherche par caractère simple. *ChaineRechAlt* fonctionne comme *ChaineRechPick* tant que le caractère entré est identique. A ce moment la chaîne de recherche est alors reinitialisée et la barre lumineuse positionnée sur le prochain élément commençant par le nouveau caractère.

Lorsque *CaracRechPick* est activée, l'entrée d'un caractère alphanumérique entraîne le déplacement de la barre lumineuse sur le prochain élément qui commence par ce caractère. *CaracPickActuel* a le même comportement et il y a sortie de la routine Pick avec la mise en correspondance du dernier choix.

Pour toutes ces options, la recherche circulaire ne tient pas compte des majuscules et des minuscules. S'il n'y a plus d'élément correspondant au-delà de l'élément courant, la recherche se poursuit en début de la liste.

Constantes

```
AttrPickAlt : Boolean = False;
```

Alors que les autres constantes doivent être initialisées avant d'appeler *FenPick*, la constante *AttrPickAlt* devra être modifiée à l'intérieur de la fonction chaîne spéciale définie pour chaque liste Pick. Chaque fois que la fonction chaîne est appelée pour renvoyer un nom d'élément, elle doit également positionner *AttrPickAlt* à "True". TPPICK affichera alors cet élément particulier en utilisant les autres couleurs définies dans *TabCouleurPick*. TPPICK réinitialise alors *AttrPickAlt* à "False" avant d'appeler la fonction chaîne.

Les constantes *AttrPickAlt* sont illustrées. L'unité TPDIR affiche les noms de sous-répertoires avec des attributs différents à partir des noms de fichiers normaux.

```
CacheCurseur : Boolean = True;
```

Par défaut, TPPICK désactive le curseur hardware clignotant lorsqu'une fenêtre Pick est affichée et le restaure lorsque la fenêtre Pick est désactivée. Si *CacheCurseur* vaut "False", TPPICK laisse la forme de curseur inchangée pendant que la fenêtre Pick est affichée.

```
LongMaxRech = 16;   {Longueur maximum de la chaîne de recherche
                     incrémentale}
```

Cette constante spécifie la longueur maximum de n'importe quelle chaîne de recherche incrémentale. Sa valeur n'a de sens que si la constante *RechPick* est égale à *ChaineRechPick*.

```
MatricePick : Byte = 1;
```

MatricePick est une constante importante que vous utiliserez souvent. Elle spécifie le nombre de colonnes dans une liste Pick. Pour les listes courtes, une seule colonne est nécessaire mais pour des listes plus longues, telles que des répertoires de fichiers, il vaudrait mieux plusieurs colonnes. Pour utiliser la constante *MatricePick*, affectez-lui la valeur souhaitée avant d'appeler *FenPick* ou n'importe quelle autre routine de TPPICK.

Remarquez que TPPICK affiche les éléments dans l'ordre, colonne par colonne : pour une liste alphabétique, tous les éléments commençant par la lettre A seront dans la première colonne, ceux commençant par la lettre B dans la seconde, etc..

```
LignesMinPick : Word = 0;
LignesMaxPick : Word = 9999;
```

LignesMinPick et *LignesMaxPick* spécifient le degré de liberté de TPPICK pour choisir la taille de la fenêtre. Bien que vous spécifiiez une hauteur de fenêtre pour TPPICK lorsque vous appelez *FenPick*, TPPICK adaptera cette hauteur s'il y a moins d'éléments qu'il y a de lignes spécifiées dans la fenêtre. TPPICK garantit toujours que le nombre de lignes dans la fenêtre sera inférieur à *LignesMaxPick* et supérieur à *LignesMinPick*, indépendamment d'autres paramètres ou calculs.

```
PickEncore : Boolean = True;
```

Lorsque la liste Pick contient plus d'élément qu'il n'est possible d'en visualiser, TPPICK affiche un message dans le cadre de la fenêtre Pick indiquant qu'il a d'autres éléments qui ne sont pas visualisés. Le message "Suite" n'apparaîtra pas lorsque *PickEncore* vaut "False".

```
DerouleSourisPick : Boolean = True;
```

Lorsque les fonctionnalités de la souris de TPPICK sont activées (en appelant *ActivePickSouris*), cette constante vérifie si TPPICK affiche et supporte une bande de défilement. Puisque la bande de défilement apparaît sur le côté droit du cadre de la fenêtre Pick, elle n'est disponible que lorsque la fenêtre comporte un cadre. De plus, la bande de défilement apparaît seulement lorsqu'il y a plus d'éléments à afficher que ne peut en contenir la fenêtre. La bande de défilement a plusieurs fonctionnalités. Dans les coins supérieurs et inférieurs du cadre, des flèches verticales sont affichées. TPPICK déroule l'affichage des éléments ligne par ligne toutes les fois que le pointeur de la souris a été positionné sur l'une de ces flèches et que le bouton gauche est pressé. TPPICK affiche également un bloc dans le cadre vertical pour indiquer la position relative de l'élément actuel dans la liste toute entière. TPPICK affiche un jeu d'élément différents et adapte ce curseur lorsque le pointeur de la souris est placé à un emplacement différent dans la bande de défilement et que le bouton gauche est pressé.

```
RechPick : TypeRech = PasRechPick;
```

Cette constante contrôle le type de recherche disponible dans la fenêtre Pick. Voir la définition de *TypeRech* ci-dessus pour plus de détails. Par défaut, il n'y a pas de recherche alphanumérique.

```
BarrePick : Boolean = True;
```

Lorsque la barre lumineuse atteint la bas de la colonne sur une liste Pick courte, la barre se déplace automatiquement sur l'élément supérieur. Pour les listes plus longues (celles qui comportent plus d'éléments que ne peut en contenir la fenêtre à un moment donné), cela entraîne le déroulement de la fenêtre en même temps que le repositionnement. Lorsque *BarrePick* vaut "True", le repositionnement est désactivé pour les listes qui doivent être déroulées. Positionnez-le à "False" pour permettre un repositionnement, même dans ce cas.

```
TPKRien = 0;     {N'est pas une commande}
TPKAlpha= 1;     {Caractère utilisé pour les recherches alphanumériques}
TPKHaut = 2;     {Curseur sur la ligne précédente}
TPKBas = 3;      {Curseur sur la ligne suivante}
TPKPgup = 4;     {Curseur sur la page précédente}
TPKPgdn = 5;     {Curseur sur la page suivante}
TPKGauche = 6;   {Curseur sur la colonne de gauche}
TPKDroite = 7;   {Curseur sur la colonne de droite}
TPKSortie = 8;   {Sortie de la liste Pick sans opérer de sélection}
TPKSelect = 9;   {Sélectionne l'élément courant et sort de la liste}
TPKAide = 10;    {Appel de la routine d'aide}
TPKHome = 11;    {Déplace la curseur sur le premier élément}
TPKEnd = 12;     {Déplace le curseur sur la dernier élément}
TPKSonde = 13;   {Contrôle la position de la souris et exécute une
                  action}
```

```
TPKUtl0 = 14;   {Commande de sortie définie par l'utilisateur}
...
TPKUlt3 = 17;
```

Codes de déplacement du curseur et de sélection des commandes de TPPICK. La plupart d'entre eux s'expliquent d'eux-mêmes. Les quatre derniers (*TPKUtl0..TPKUtl3*) permettent de définir des commandes utilisateurs. Si l'une de ces commandes est exécutée, TPPICK se comportera de la même façon que si vous aviez pressé (ENTER). Il vous appartient alors de contrôler la variable globale *NumCmdPick* et d'exécuter la fonction associée avec la commande demandée. Pour plus de détails sur l'implémentation des commandes définies par l'utilisateur, voir l'appendice C.

```
DebutRech : Byte = 1;
```

Lorsqu'une méthode de recherche alphanumérique est activée, cette option cherche où commence la recherche dans chaque chaîne d'éléments. Par défaut, la recherche commence sur le premier caractère de chaque élément.

Variables

```
CaracPick : char;
```

Dernière touche entrée dans TPPICK. Pour les touches étendues, *CaracPick* contient le code de balayage de la touche OR $80 ce qui correspond au bit de poids fort égal à 1. Remarquez que *CaracPick* contiendra la touche réelle entrée lorsque l'option *CaracPickActuel* est sélectionnée et qu'une touche alphanumérique a sélectionné un élément donné.

```
NumCmdPick : TPType;
```

Dernière commande donnée à TPPICK. Peut être utilisée pour déterminer si l'utilisateur a sélectionné un élément (*NumCmdPick* = *TPKSelect*) ou est sorti sans opérer de sélection (*NumCmdPick* = *TPKSortie*). Remarquez que la variable *NumCmdPick* sera égale à *TPKSelect* lorsque l'option *CaracPickActuel* est sélectionnée et lorsqu'une touche alphanumérique a été entrée pour sélectionner un élément donné. Il est préférable de contrôler *NumCmdPick* plutôt que d'utiliser *CaracPick* si les touches de chaque commande ont été reaffectées.

```
PtrAidePick : Pointer;
```

Pointe sur une routine à appeler lorsque la commande d'aide est donnée. Voir l'appendice C pour plus de détails.

```
PtrTouchePick :Pointer;
```

Pointe sur une routine qui renverra la prochaine touche. Voir l'appendice C pour plus de détails.

```
PtrUtlPick : Pointer;
```

Pointe sur une routine qui sera exécutée une fois que l'écran aura été mis à jour chaque fois qu'une commande clavier Pick a été entrée. Remarquez que l'utilisation de cette routine est facultative. Elle ne correspond pas à la fonction chaîne demandée dont l'adresse est passée comme paramètre à *FenPick*. Si cette variable est différente de nil, elle doit pointer sur une procédure compilée avec le modèle FAR et sous la forme suivante dans laquelle *Elem* est l'index de l'élément sélectionné actuellement :

```
procedure UtlPick(Elem : Word);
```

PtrUtlPick est utile lorsque le déplacement de la barre lumineuse Pick pourrait entraîner la modification d'autres écrans en dehors de la fenêtre Pick elle-même.

ActivePickSouris

Déclaration

```
procedure ActivePickSouris;
```

Objet

Active la souris pour les listes Pick

Commentaires

Appelez cette routine pour activer la souris dans les listes Pick. La routine vérifie si la souris est installée, puis initialise un indicateur pour que les autres routines Pick puissent utiliser la souris. Vous devrez appeler *ActivePickSouris* avant d'appeler *FenPick* de TPPICK.

Puisque le contrôle de la souris de TPPICK dépend du gestionnaire d'événements de souris de TPSOURIS, *ActivePickSouris* appelle également la routine *ActiveAttenteEvenement* de TPSOURIS. Aucun problème si vous avez déjà appelé cette routine. Sachez cependant que tout gestionnaire d'événements de souris activé précédemment perdra le contrôle après l'appel de *ActivePickSouris*.

ActivePickSouris ne modifie pas la fenêtre souris actuelle, le type du pointeur de la souris ou le ratio Mickey-en-pixels, pas plus qu'elle n'affiche le pointeur de la souris. Votre programme devra initialiser ces valeurs en appelant les routines appropriées de TPSOURIS. Vous pourrez utiliser un pointeur de souris software ou un pointeur hardware.

Lorsque la souris est activée, TPPICK exécute les actions suivantes pendant les sélections de liste Pick. Lorsque le bouton gauche de la souris est libéré, TPPICK déplace la barre lumineuse sur l'élément où se trouve actuellement le pointeur de la souris. La libération du bouton gauche une seconde fois avec le pointeur de souris sur l'élément mis en valeur, entraîne la sortie de la routine Pick. La sélection par double clic permet un meilleur contrôle par l'utilisateur. Le pression du bouton droit dans une fenêtre Pick est équivalente à la frappe de la touche (ESC) et entraîne la sortie de la fenêtre sans opérer de sélection. La pression des deux boutons est équivalente à la frappe de (F1) et active une routine d'aide définie par l'utilisateur éventuellement.

Si la fenêtre Pick comporte un cadre et que la constante typée *DerouleSourisPick* de TPPICK vaut "True", la souris peut être également utilisée pour dérouler les listes Pick qui sont plus longues que la fenêtre. Voir la description de *DerouleSourisPick* pour plus de détails.

Les actions mises en oeuvre lorsque les boutons de la souris sont activés, peuvent être configurées comme les touches. Voir l'appendice C pour plus de détails.

Remarquez que la souris est mise en oeuvre lorsque les boutons sont relâchés et non lorsqu'ils sont pressés. Cette technique permet une utilisation fiable des commandes qui demandent la pression des deux boutons simultanément. Vous pouvez modifier cette fonctionnalité en basculant la constante *AttenteRelacheBouton* de TPSOURIS. Après cette modification, les fonctions de la souris s'exécuteront lorsque le bouton de la souris sera relâché.

Dans un programme résident utilisant TPPICK, n'activez pas la souris avant de lancer le programme résident. Activez la souris après le lancement du programme et désactivez-la avant le retour.

D'autres considérations sont à prendre en compte lors de l'écriture des programmes résidents qui utilisent la souris. Voir au chapitre 2, la partie sur TPSOURIS pour plus de détails.

Exemple

```
begin
   ClrScr;
   WriteLn('Choisissez une garniture');
   TpPick.RechPick := CarPickActuel;       Accepte la sélection sur la
                                            première lettre du mot}
   TpPick.DerouleSourisPick := True;       {Déroulement avec la souris}
   ActivePickSouris;        {Active la souris dans la fenêtre Pick}
   Choix := 1;                        {Choix initial}
   if not FenPick(@GarniturePizza,  {Retourne les noms d'éléments}
            NumGarniturePizzas,     {Nombre d'éléments dans la liste
                                     Pick}
            5, 5, 17, 11,           {Coordonnées de la fenêtre}
            True,                   {Dessine un cadre autour de la
                                     liste}
            CouleursPick,           {Couleurs à utiliser}
            '',                     {Pas de titre pour cette fenêtre}
            Choix)                  {Elément choisi}
   then             {Mémoire insuffisante pour le Buffer d'écran}
      WriteLn('Impossible d''exécuter FenPick');
   else
      if NumCmdPick = TPKSelect then
         {Un choix a été fait}
         WriteLn('Garniture choisie : ',GarniturePizza(Choix))
      else
         {Escape pressé}
         WriteLn('Aucun choix');
end.
```

Affiche les deux lignes de modification dans l'exemple de l'introduction nécessaires pour activer toutes les fonctions de la souris.

Voir aussi

DesactivePickSouris (TPPICK) ActiveAttenteEvenement (TPSOURIS)

ChoixPick

Déclaration

```
procedure ChoixPick(F : PtrFen;
                    FoncChaine : pointer;
                    NbElems : Word;
                    Couleur : TabCouleurPick;
                    EffaceBarre : Boolean;
                    Choix : Word;
                    Ligne : Word);
```

Objet

Sélection dans une liste Pick déjà affichée.

Commentaires

ChoixPick est voisine de *RemplitFenPick* mais avec des fonctionnalités supplémentaires. Lorsque *EffaceBarre* vaut "True", l'élément sélectionné reprend sa couleur initiale.

Choix et *Ligne* servent à initialiser la sélection mise en valeur au départ. Lorsque *ChoixPick* se termine, *Choix* contient l'index de l'élément sélectionné et *Ligne* contient la ligne dans la fenêtre relative où cet élément était affiché. La dernière valeur servira lors d'un appel ultérieur à *ChoixPick* et ne devrait généralement pas être traitée par le programme appelant.

Vérifiez que les valeurs des constantes de recherche de TPPICK (*RechPick*, etc...) ont été initialisées avant d'appeler *ChoixPick*.

ChoixPick et *FenPick* sont des programmes ré-entrants. Si la sélection d'un élément conduit à l'affichage d'une autre liste Pick, vous pouvez en toute sécurité changer les valeurs de n'importe quelle constante globale de TPPICK avant d'afficher la seconde liste Pick. TPPICK la sauvegarde et la restaure automatiquement en entrée et en sortie. Il vous appartient d'effacer et de libérer chaque fenêtre allouée par *RemplitFenPick* avant de revenir à une liste Pick précédente.

Exemple

Voir l'exemple de *RemplitFenPick*.

Voir aussi

RemplitFenPick (TPPICK)

DesactivePickSouris

Déclaration

```
procedure DesactivePickSouris;
```

Objet

Désactive le contrôle des listes Pick par la souris.

Commentaires

Cette routine annule les effets de *ActivePickSouris*. Dans la plupart des programmes, vous ne devrez jamais appeler *DesactivePickSouris* puisque ses fonctions principales sont gérées automatiquement par la routine de sortie de *Turbo Pascal*. Dans certains cas, et particulièrement dans les programmes résidents, vous pourrez appeler *DesactivePickSouris* afin de désactiver temporairement les fonctions de la souris.

Remarquez que *DesactivePickSouris* ne cache pas le pointeur de la souris. Appelez *CacheSouris* pour cela.

Exemple

```
{Activation TSR}
...
ActivePickSouris;
if FenPick(...) then begin
   ...
end;
DesactivePickSouris;
...
{Désactivation TSR}
```

Voir aussi

ActivePickSouris (TPPICK)

FenPick

Déclaration

```
function FenPick(FoncChaine : pointer;
                 NbElems : Word;
                 XBas, YBas : Byte;
                 XHaut, YHaut : Byte;
                 DessinCadre : Boolean;
                 Couleurs : TabCouleurPick;
                 Entete : String;
                 var Choix : Word) : Boolean;
```

Objet

Affiche une fenêtre Pick, permet à l'utilisateur de la balayer et renvoie l'élément sélectionné.

Commentaires

FenPick combine les fonctions de *RemplitFenPick* et *ChoixPick* en une seule opération. *FenPick* définit une fenêtre, l'affiche, la remplit, permet à l'utilisateur de "browser" la liste et renvoie l'élément sélectionné.

FoncChaine est l'adresse d'une fonction qui renvoie le nom de chaque élément Pick. Cette fonction doit être compilée sous le modèle FAR et déclarée avec la syntaxe suivante :

```
{$F+}                          {Doit être FAR}
function NomElement(Element : Word) : string;
   {Retourne le nom des éléments}
begin
   NomElement := ...           {Doit retourner un nom d'élément}
   AttrPickAlt := ...          {Facultatif}
end;
{$F-}
```

Chaque fois que *FenPick* affiche un élément, elle appelle la fonction chaîne spécifiée pour lire le nom de l'élément. *FenPick* passe à la routine un index dans l'intervalle 1..*NbElems*. La chaîne renvoyée par la fonction chaîne est affichée non modifiée avec une exception : des blancs sont ajoutés pour étendre la chaîne à la longueur voulue. Outre le renvoi du nom de l'élément, la fonction chaîne peut initialiser une autre variable qui modifiera la présentation de l'affichage. Si la constante de TPPICK, *AttrPickAlt*, vaut "True", l'élément sera affiché en utilisant les autres couleurs (décrites dans *TypeCouleurPick* ci-dessus).

Comme la fonction chaîne est appelée assez souvent pendant le balayage de la liste, nous vous recommandons d'effectuer le moins de calculs possibles à chaque appel. Si possible, faites un pré-traitement sur les chaînes à renvoyer. Voir TPDIR en exemple : le répertoire de fichiers est lu une seule

fois et les chaînes contenant les noms formatés, les dates, les heures et les tailles sont générées une seule fois avant d'appeler *FenPick*.

XBas, YBas, XHaut, YHaut définissent les coordonnées de la fenêtre où la liste Pick apparaîtra. Ces valeurs comportent un espace pour le cadre de la fenêtre si *DessinCadre* vaut "True". La largeur disponible pour l'affichage de chaque élément Pick est définie (approximativement) par *(XHaut-XBas) div MatricePick* où *MatricePick* est le nombre de colonnes des éléments. Il vous appartient de spécifier des valeurs de *XBas, XHaut* et *MatricePick* suffisantes pour chaque élément à afficher. TPPICK tronquera ou complétera chaque nom d'élément pour qu'il se cadre correctement dans l'espace alloué.

FenPick peut également adapter la valeur de *YHaut* spécifiée. Si le nombre d'éléments Pick est plus petit que le nombre de lignes disponibles en *XBas* et *YHaut, FenPick* réduira *YHaut*. Si le nombre de lignes disponibles dépasse *LignesMaxPick* ou est inférieur à *LignesMinPick,* elle adaptera *YHaut*. Si le nombre d'éléments Pick dépasse le nombre de lignes écran, TPPICK activera un déroulement vertical.

L'entête, s'il existe, apparaîtra centré en haut du cadre de fenêtre.

Choix spécifie quel élément sera mis en valeur initialement, et contiendra à la sortie de *FenPick* l'index de l'élément sélectionné.

FenPick renvoie "True" si la fenêtre a été affichée correctement et si les paramètres passés à la fonction sont corrects. Elle renvoie "False", en cas d'erreur programme, en cas d'espace insuffisant sur la tas ou si des paramètres incorrects ont été passés. *FenPick* suppose que la place réservée sur le tas est suffisante pour contenir la zone de l'écran recouverte par la fenêtre plus un peu de mémoire pour la pile de la fenêtre.

Si la fonction *FenPick* se termine bien, les variables globales de TPPICK, *NumCmdPick* et *CaracPick,* contiendront respectivement la commande utilisée pour sortir de la fonction Pick et la touche finale. Il est préférable de contrôler la valeur *NumCmdPick* plutôt que *CaracPick* puisque les touches peuvent être réinstallées avec des valeurs différentes. Les valeurs suivantes sont retournées dans *NumCmdPick* à la sortie de *FenPick* :

```
TPKSortie    {Sortie de la liste Pick sans sélection}
TPKSelect    {Sélectionne l'élément courant et sort de la liste Pick}
TPKUtl0      {Commande de sortie définie par l'utilisateur}
....
TPKUtl3
```

L'unité TPAIDE utilise la commande de sortie définie par l'utilisateur, *TPKUtl3*, pour activer la fonction "Thème d'aide précédent" lors de la recherche dans l'index d'aide. Vous éviterez de redéfinir *TPKUtl3* si votre application utilise aussi TPAIDE.

Vérifiez que les valeurs des constantes de TPPICK (*MatricePick, RechPick,* etc...) ont été initialisées avant d'appeler *FenPick*.

Remarquez que les options de TPFENETR (*Ombre, Explosion,* etc..) actives au moment de l'appel à *FenPick* modifieront la présentation de la fenêtre Pick.

Exemple

Voir au début de cette section un exemple simple mais complet utilisant *FenPick*. Voir les unités TPDIR et TPAIDE ainsi que le programme de démonstration SAISIE pour des cas plus complexes.

Voir aussi

RemplitFenPick (TPPICK) ChoixPick (TPPICK)
InitFen (TPFENETR)

RemplitFenPick

Déclaration

```
procedure RemplitFenPick(F: PtrFen;
                         FoncChaine : pointer;
                         NbElems : Word;
                         Couleur : TabCouleurPick;
                         Choix : Word;
                         Ligne : Word);
```

Objet

Affiche une fenêtre prédéfinie, la remplit avec la variable *Choix* et revient au programme appelant.

Commentaires

RemplitFenPick met en oeuvre une partie de la routine principale *FenPick*. *RemplitFenPick* est généralement utilisée avec *ChoixPick* décrite plus loin. Cette combinaison est préférable à *FenPick* lorsque plusieurs sélections doivent être faites à partir d'une liste Pick simple. Alors que *FenPick* dessine et efface la fenêtre Pick à chaque appel, *ChoixPick* utilise une fenêtre initialisée précédemment par *RemplitFenPick* et laisse la fenêtre sur l'écran lors de la sortie. Ceci évite un léger scintillement de l'écran qui apparaîtrait si *FenPick* était appelé de façon répétitive.

La fenêtre *F* doit être initialisée en appelant *InitFen* (TPFENETR) avant d'appeler *RemplitFenPick*. La fenêtre *F* doit être déjà sur l'écran au moment où *RemplitFenPick* est appelée et doit être la fenêtre actuelle. Si *F* n'est pas active, *RemplitFenPick* l'affichera.

FoncChaine est l'adresse d'une fonction qui renverra le nom de chaque élément Pick. Voir l'introduction de cette partie et l'entrée de *FenPick* pour plus de détails. *NbElems* spécifie le nombre d'éléments de la liste Pick indexée de 1 à *NbElems*. *Couleur* spécifie les couleurs à utiliser pour la liste Pick.

Choix spécifie l'élément à mettre en valeur initialement et *Ligne* spécifie la ligne relative mise en valeur dans la fenêtre. Normalement, ces deux paramètres doivent avoir la valeur 1 ; sauf s'il y plus d'éléments que la fenêtre ne peut en contenir, la valeur de *Ligne* est superflue et sera initialisée par *Choix*.

Remarquez que *RemplitFenPick* modifie l'écran et retourne immédiatement au programme appelant sans l'attente de la frappe des touches. Elle laisse la fenêtre spécifiée active et les coordonnées du curseur restent dans les limites de la fenêtre Pick.

Vérifiez que les valeurs de toutes les constantes d'affichage de TPPICK (*MatricePick*, etc...) sont initialisées avant d'appeler *RemplitFenPick*.

Exemple

```
var
   F : PtrFen;
   Choix : Word;
   Ligne : Word;
begin
   {Définition de la fenêtre}
   if not InitFen(F, 5, 5, 17, 11, True, False, False,
            CouleursPick[AttrFen], CouleursPick[AttrCadre],
            CouleursPick[AttrEntete], '') then Exit;

   {Eléments de la fenêtre}
   Choix := 1;
   Ligne := 1;
   RemplitFenPick(F, @GarniturePizza, NumGarniturePizzas,
         CouleursPick, Choix, Ligne);

   {Lecture des choix tant que Esc n'est pas pressé}
   repeat
      ChoixPick(F, @GarniturePizzas, NumGarniturePizzas,
               CouleursPick, False, Choix, Ligne);
      if NumCmdPick = TPKSelect then
         {Un choix a été fait}
         EcritVite('Le dernier choix était '
                  +Complete(GarniturePizza(Choix), 20), 15, 1, $07);
   until NumCmdPick <> TPKSelect;

   {Efface et dispose de la fenêtre}
   DealloueFen(EffaceFenSup);
end.
```

Variante de l'exemple présenté au début de cette section montrant comment *ChoixPick* peut être utilisée pour sélectionner plusieurs choix dans une fenêtre. Tout d'abord, la fenêtre est définie par un appel à *InitFen*, ensuite *RemplitFenPick* dessine la fenêtre avec un ensemble initial de choix. *ChoixPick* est appelée répétitivement et la sélection courante est affichée sur l'écran. Lorsque l'utilisateur presse (ESC), la boucle se termine, la fenêtre est effacée et libérée.

Voir aussi

InitFen (TPFENETR)

ChoixPick (TPPICK)

TPDIR : Répertoires "popup"

TPDIR utilise l'unité TPPICK pour résoudre un problème de programmation courant : la sélection de noms de fichiers. La seule routine de TPDIR, *LitNomFic*, accepte en entrée des données avec jokers, ouvre une fenêtre contenant les fichiers correspondants, permet à l'utilisateur de les parcourir et renvoie le nom du fichier sélectionné.

En tirant profit de la souplesse inhérente à TPFENETR, TPPICK et TPDATE, TPDIR offre toutes les possibilités voulues. La fenêtre répertoire peut avoir n'importe quelle forme, taille et couleur ; elle pourra exploser ou imploser à la demande. Les fichiers peuvent être affichés sur une ou plusieurs colonnes avec ou sans date, heure et taille, dans un format que vous pouvez spécifier. Dans la liste pick, les fichiers peuvent être sélectionnés par leur première lettre ou en utilisant une recherche incrémentale. TPDIR permet même à l'utilisateur de parcourir les multiples sous-répertoires en pointant le nom du répertoire et en pressant une touche.

L'utilisation de TPDIR est facile comme le démontre l'exemple suivant :

```
program ExempleDir;
uses
   Dos, TpChaine, TpCrt, TpFenetr, TpCmd, TpPick, TpDir;
const
   CouleursDir : TabCouleurPick = ($07, $07, $0F, $70, $0F, $78);
var
   Masque : string[79];
   NomF : string[79];

begin
   ClrScr;

   TpPick.RechPick := ChaineRechPick;      {Active la recherche
                                            incrémentale}
   TpDir.AffTailleDateHre := True;         {Affiche les taille, Date et
                                            heure des fichiers}
```

```
    repeat
       Write('Entrez un masque de nom de fichier : ');
       ReadLn(Masque);
       case LitNomFic(Masque,        {Masque de nom de fichier}
                      AnyFile,       {Tout type de fichier (défini dans
                                      l'unité DOS)}
                      1, 3,          {Position de la fenêtre de
                                      répertoire}
                      15, 1,         {Utilise une colonne se terminant
                                      en ligne 15}
                      CouleursDir,   {Spécifie les couleurs à utiliser}
                      NomF           {Nom de fichier choisi}
                      ) of
          0 : if Length(NomF) = 0 then
                WriteLn('Vous avez pressé Esc')
              else
                WriteLn('Vous avez choisi ',NomF);
          1 : WriteLn('Chemin introuvable');
          2 : WriteLn('Aucun fichier ne correspond');
          3 : WriteLn(NomF,' est un nouveau fichier');
          4 : WriteLn('Mémoire insuffisante');
       else
          WriteLn('Erreur DOS critique');
       end;
    until Length(NomF) = 0;
  end.
```

CouleursDir a le même type que le tableau de couleurs de TPPICK. Bien qu'une seule valeur soit spécifiée ici, vous pourrez normalement avoir deux tableaux de couleurs, l'un pour un moniteur couleur et l'autre pour un moniteur monochrome.

Masque est une chaîne de caractères jokers du DOS pour les fichiers que *LitNomFic* doit rechercher. *NomF* est le fichier sélectionné, *LitNomFic* renvoie toujours un nom de chemin complet comportant un numéro du lecteur, un répertoire, un nom et un suffixe.

Au début du programme, nous initialisons deux constantes pour personnaliser TPPICK et TPDIR. Dans l'exemple, nous activons une recherche incrémentale avec TPPICK et affichons la taille du fichier, la date et l'heure avec TPDIR.

Ensuite, nous activons une boucle pour lire le masque du fichier puis utilisons *LitNomFic* pour activer une fenêtre répertoire. *LitNomFic* renvoie un code d'erreur que nous pouvons tester.

TPDIR : Déclarations

Constantes

```
FormatDate : string[12] = 'Mm/jj/aa';
```

Cette chaîne est un format d'affichage de la date (comme définie dans TPSAISIE et TPDATE). Si la chaîne est vide, la date ne s'affichera pas, sinon les caractères du masque défini dans TPDATE peuvent être combinés pour formater la date comme souhaité. Avec le format date par défaut, les dates apparaîtront comme suit :

```
2/02/88
12/29/87
```

```
DirsEnMajuscules : Boolean = True;
```

Si cette constante vaut True, les noms des sous-répertoires sont affichés en caractères majuscules sinon en minuscules.

```
AffichageDir : string[5] = '<rep>';
```

Cette chaîne est affichée à la place de la taille du fichier pour les entrées de sous-répertoires.

```
FichiersEnMajuscules : Boolean = False;
```

Si cette contante vaut "True", les noms des fichiers sont affichés en caractères majuscules, sinon en minuscules.

```
DirsSepares : Boolean = True;
```

Si cette constante vaut "True", les noms des sous-répertoires sont triés, sinon ils sont affichés dans l'ordre alphabétique, mélangés aux fichiers.

```
AffSuffixe : Boolean = True;
```

Si cette constante vaut "True", les suffixes de fichiers sont affichés. Cette constante sera initialisée à False seulement si la fenêtre répertoire doit contenir un groupe de fichiers ayant tous le même suffixe.

```
AffTailleDateHre : Boolean = False;
```

Si cette constante vaut "True", la taille, la date et l'heure de création des fichiers doivent être affichées dans la fenêtre répertoire en fonction des constantes décrites ci-dessous. Si elle vaut false, seul le nom du fichier est affiché.

```
AffTaille : (TailleRien,          {N'affiche pas la taille fichier}
            TailleOctets,         {Affiche la taille en octets}
            TailleKOctets)        {Affiche la taille en Ko}
          = TailleOctets;
```

La taille peut être affichée en octets ou arrondie au nombre de Ko le plus proche.

```
FormatHeure : string[12] = 'HH:mmt';
```

Cette chaîne est le format d'affichage de l'heure. Si la chaîne est vide, l'heure ne sera pas affichée. Sinon les caractères du masque défini dans TPDATE peuvent être combinés pour formater l'heure comme souhaité. Avec la valeur par défaut de *FormatHeure*, les heures sont affichées comme suit :

```
6:12a
11:55p
```

```
UtiliseCadreFic : Boolean = True;
```

Si cette constante vaut True, un cadre est dessiné autour de la fenêtre répertoire en utilisant le cadre courant enregistré dans *CaracsCadre* (TPCRT).

LitNomFic

Déclaration

```
function LitNomFic(Masque : string;
                   AttrFichier : Byte;
                   XBas, YBas : Byte;
                   YHaut, ColsPick : Byte;
                   Couleurs : TabCouleurPick;
                   var NomFichier : String): Word;
```

Objet

Etant donné un masque (qui peut contenir des jokers), cette fonction affiche une fenêtre répertoire, permet à l'utilisateur de faire son choix et renvoie un nom de chemin. Renvoie un code d'erreur.

Commentaires

Si vous passez une chaîne vide, *LitNomFic* la considère comme égale à '*.*' et active une fenêtre répertoire. Si vous passez un nom de fichier complet, *LitNomFic* vérifie si le fichier existe et renvoie un code d'erreur sans activer la fenêtre. Si vous passez le nom d'un lecteur ou d'un sous-répertoire, *LitNomFic* affiche une fenêtre contenant tous les fichiers. Les règles de spécification de *Masque* doivent correspondent à celles des jokers du DOS.

AttrFichier définit les fichiers à rechercher. Les valeurs possibles pour *AttrFichier* sont définies dans l'unité DOS du *Turbo Pascal*. Si le bit "Directory" ($10) de *AttrFichier* est positionné, *LitNomFic* affichera les sous-répertoires dans sa liste et permettra à l'utilisateur de sélectionner des sous-répertoires pour parcourir le disque dur. Si le bit "Directory" n'est pas positionné, seuls les fichiers normaux seront affichés et l'utilisateur ne pourra parcourir qu'un seul répertoire. En sélectionnant une valeur appropriée de *AttrFichier*, vous pouvez faire apparaître lors de l'affichage du répertoire, les fichiers en lecture seule, les fichiers cachés et les fichiers système.

XBas et *YBas* spécifient le coin supérieur gauche de la fenêtre répertoire, y compris le cadre s'il est activé. *ColsPick* détermine le nombre de colonnes des noms des fichiers dans la fenêtre pick. Si la taille, la date et l'heure sont affichées, vous devez essayer de cadrer deux colonnes de noms de fichiers sur l'écran. Sinon, vous pourrez avoir au plus cinq colonnes. Si vous spécifiez une valeur de *ColsPick* inadaptée, *LitNomFic* l'ajustera pour vous. *YHaut* spécifie la plus haute ligne de la fenêtre répertoire. Cette valeur est adaptée par TPPICK. Voir cette section pour plus de détails.

Couleurs définit les attributs vidéo du répertoire. Les attributs vidéo sont affichés avec les attributs associés à *AttrFen* et *AttrSelect* (s'ils sont

sélectionnés). Les sous-répertoires sont affichés en utilisant *AltNormal* et *AltHaut* (s'ils sont sélectionnés).

NomFichier renvoie le nom de chemin complet du fichier sélectionné. Si une erreur se produit dans *LitNomFic* ou si l'utilisateur presse (ESC), le nom de fichier sera une chaîne vide.

LitNomFic renvoie un code d'erreur en résultat. Voici les valeurs :

0	Succès
1	Chemin non trouvé
2	Pas de fichier correspondant
3	Nouveau fichier
4	Mémoire insuffisante
5	Ne tient pas sur l'écran.

ou un code d'erreur critique du *Turbo Pascal*

Le code 1 signifie qu'un lecteur non valide ou un nom de sous-répertoire non valide ont été spécifiés dans le masque d'entrée. Un code 2 signifie qu'il n'y a pas de fichier qui corresponde au masque spécifié et à l'attribut. Le code 3 signifie qu'un nom de fichier complet valide a été donné dans le masque d'entrée mais qu'aucun fichier de ce nom n'existe actuellement. Un code 4 signifie qu'il n'y pas assez de place sur le tas pour le fonctionnement interne de *LitNomFic*. Un code 5 signifie que les coordonnées de la fenêtre spécifiée sont incorrectes ; même une seule colonne de noms de fichiers ne pourra être affichée avec ces coordonnées. Les autres valeurs différentes de 0 du code d'erreur correspondent à des erreurs critiques de *Turbo Pascal*. Par exemple, ces valeurs seront renvoyées après tentative de lecture d'un fichier sans disquette dans le lecteur.

TPDIR utilise le tas pour deux raisons. Premièrement, TPDIR alloue de la place sur la tas pour les buffers de fenêtre. La quantité de mémoire nécessaire est fonction principalement de la taille de la fenêtre. Deuxièmement, elle alloue temporairement de la place pour enregistrer un tableau de noms de fichiers. La place par fichier dépend de la taille, de la date et de l'heure - elle est comprise entre une valeur minimum de 16 octets pour des fichiers sans date, heure ou taille, et un maximum de 50 octets pour des fichiers spécifiés avec masque. TPDIR allouera de la place pour au plus 500 fichiers dans un répertoire. Si la place n'est pas suffisante, elle allouera ce qu'elle pourra et ignorera tous les fichiers en trop. *LitNomFic* s'arrêtera avec un code d'erreur égal à 4, seulement si TPDIR ne peut allouer que deux noms de fichiers au plus.

LitNomFic affiche toujours les noms de fichiers triés alphabétiquement. En fonction de l'état de *DirsSepares*, les noms de sous-répertoires seront mélangés ou séparés des fichiers normaux.

LitNomFic enregistre l'état de la constante globale *MatricePick* de TPPICK avant de la modifier et la restaure avant la sortie.

Exemple

```
Statut := LitNomFic('*.DAT',0,XBas,YBas,YHaut,1,Couleurs,NomFichier);
```

Affiche tous les fichiers normaux du répertoire courant avec le suffixe .DAT. Les sous-répertoires ne sont pas affichés ce qui interdit à l'utilisateur d'examiner les autres répertoires.

```
Statut := LitNomFic('*.DAT', Directory, XBas, YBas, YHaut, 1,
                    Couleurs, NomFichier);
```

Affiche tous les fichiers normaux du répertoire courant avec le suffixe .DAT. Les sous-répertoires sont affichés ainsi l'utilisateur peut parcourir tous les autres répertoires s'il le souhaite.

Voir l'exemple au début de cette section pour le fonctionnement complet d'un programme qui utilise TPDIR. Voir le programme de démonstration GENETAT pour l'utilisation de TPDIR dans des cas plus complexes.

Voir aussi

Formats de date et d'heure (TPDATE)

FenPick (TPPICK)

TPECRAN : Ecrans virtuels

L'objet principal de l'unité TPECRAN est de pouvoir traiter les écrans virtuels. En liaison avec l'unité TPCRT, les routines de TPECRAN vous permettent de traiter n'importe quelle région de mémoire comme un écran, d'y écrire et de déplacer son contenu sur l'écran actuel. Normalement, un écran virtuel est une page vidéo (sur un adaptateur graphique couleur) ou un buffer alloué sur le tas.

TPECRAN permet de définir jusqu'à 10 (*EcransMax*) écrans virtuels à n'importe quel moment, non compris l'écran courant, et le numéro que vous affectez à un écran permettra de le repérer (l'écran courant est toujours l'écran n° 0). Vous pouvez définir un écran de deux manières, la méthode classique est d'appeler *AlloueEcran* avec un identificateur et le nombre de lignes et de colonnes sur l'écran et de lui permettre d'allouer ou de libérer le buffer pour vous. Si vous voulez gérer vous-même l'espace mémoire, vous pouvez appeler *DefinitEcran* en lui passant un pointeur sur le buffer. Par exemple, ceci vous permettra d'utiliser plusieurs pages vidéo pour les buffers sur un système couleur et d'allouer de la place sur le tas seulement lors de l'utilisation d'un système monochrome. Souvenez-vous que n'importe quelle entrée/sortie écran faite pendant une instruction *Read* et *Readln* est toujours dirigée sur la page visible (voir ci-dessous) de l'écran courant si bien que l'utilisateur peut voir ce qu'il vient d'entrer.

Si votre programme doit être exécuté seulement sur un système couleur, vous avez une autre option. TPECRAN fait une distinction entre la *page active* et la *page visible*. La page visible est celle qui apparaît sur le moniteur, la page active est celle sur laquelle les entrées/sorties écran sont réellement dirigées. Considérons la séquence de code suivante :

```
InitPageActive(1);
ClrScr;
EcritRapide('Bonjour à tous !');
InitPageVisible(PageActive); { = 1}
```

La première ligne indique à TPECRAN d'envoyer toute sortie sur écran sur la page vidéo 1. Les seconde et troisième lignes effacent cette page et y écrivent une chaîne de caractères, et la quatrième ligne fait de la page active la page visible.

Une autre caractéristique remarquable de TPECRAN est l'adaptation des couleurs. L'un des problèmes les plus classiques dans la conception des écrans est d'assurer la correspondance entre la visualisation sur un écran couleur et sur un moniteur monochrome ou mixte (noir et blanc). Les versions spéciales de *TextColor* et *TextBackground* de TPECRAN tentent de résoudre ce problème en adoptant les combinaisons de couleurs aux combinaisons noir et

blanc lorsque votre programme fonctionne sur un système sans moniteur couleur (voir également la routine *FixeCouleur* dans TPCRT).

De bons exemples d'utilisation des routines principales de TPECRAN se trouvent dans les exemples PILEFENT.PAS et EXPLFENT.PAS. Ils démontrent à la fois la rapidité des écrans virtuels de TPECRAN et les avantages de l'adaptation des couleurs.

Restrictions importantes

Lorsque vous utilisez TPECRAN dans un programme, mais surtout lorsque vous l'intégrez dans un programme existant, n'oubliez pas ce qui suit :

- Tout d'abord, TPECRAN n'est pas magique. Elle ne peut diriger arbitrairement toutes les entrées/sorties écran vers un écran virtuel ou la page vidéo active. Les routines d'écriture écran de TPCRT, *EcritVite, EcritFacile,* etc... écriront toujours sur l'écran physique et sur la page vidéo visible. Si vous désirez écrire sur des écrans virtuels sans visualiser les pages, vous devez utiliser Write/Writeln ou l'une des routines de TPECRAN.
- Le deuxième point découle directement du premier : lorsque vous utilisez TPECRAN dans un programme, vous devrez placer une instruction *"uses TpEcran"* dans toutes les unités qui exécutent des entrées/sorties écran et TPECRAN devra apparaître après TPCRT dans la liste *uses*. Ceci parce que TPECRAN doit dupliquer de nombreuses routines de base de TPCRT (*Window, ClrScr, GotoXY,* etc...) et il est impératif que ces routines soient appelées plutôt que celles de même nom définies dans l'unité TPCRT.

Quand ne faut-il pas utiliser TPECRAN ?

De nombreux utilisateurs ont eu des problèmes liés à l'utilisation de TPECRAN alors qu'ils n'en avaient besoin. Deux règles s'imposent :

- N'utilisez pas TPECRAN si vous n'avez pas besoin d'écran virtuel.
- N'essayez pas d'utiliser TPECRAN , si vous utilisez TPFENETR ou TPMENU.

Comme TPFENETR et TPMENU, TPECRAN complète TPCRT et offre des possibilités et des fonctions supplémentaires. Contrairement aux deux autres unités, elle assure le contrôle de base des entrées/sorties écran indépendamment de TPCRT. TPECRAN gère toutes les instructions *Writeln* et *Write* aussi bien que les fonctions de base telles que le déplacement du curseur et plus précisément le fenêtrage. TPFENETR et TPMENU (qui dépend de TPFENETR) suppose que TPCRT contrôle ces fonctions et de ce fait ne peuvent être utilisées avec TPECRAN. Pour contourner cette règle, TPECRAN contrôle maintenant la présence de TPFENETR et

réciproquement. Si TPFENETR est présente dans un programme qui utilise TPECRAN, le programme ne sera pas exécuté.

Nous avons appris que plusieurs utilisateurs de *Turbo Professionnel* 4.0 utilisaient TPECRAN, seulement parce qu'ils avaient besoin des routines *EnregistreCoordsFen* et *RestaureCoordsFen*. Pour cette raison, nous avons ajouté ces routines (ainsi que la déclaration de type pour *CoordFen*) à TPCRT. Ces routines sont fonctionnellement identiques à celles de TPECRAN. Nous avons également ajouté d'autres routines dans TPCRT et TPFENETR pour vous éviter d'utiliser TPECRAN lorsque cela n'est pas nécessaire.

Routines appartenant également à l'unité CRT de Turbo

En général, les routines qui suivent sont fonctionnellement identiques aux routines de même nom de l'unité CRT de *Turbo Pascal*. Les seuls points nouveaux qui méritent d'être notés sont :

1. *ClrScr* et *ClrEol* peuvent utiliser à la fois un caractère de fond autre que ' '. Voir les explications de *InitCarFond*.
2. Si la constante typée globale *Turbo3StyleCouleurs* vaut "True", *HighVideo*, *NormVideo*, et *LowVideo* se comportent comme en Turbo 3.x
3. Comme nous l'avons vu, *TextColor* et *TextBackground* tentent d'adapter les couleurs pour améliorer la présentation sur les écrans monochromes.
4. Lorsque l'écran actif est un écran virtuel, vous ne pourrez pas créer une fenêtre plus grande que les dimensions de l'écran virtuel avec la procédure *Window*.

```
Procedure ClrScr;
   {efface la fenêtre active, utilise le caractère spécial de fond,
    sinon ' '}

Procedure ClrEol;
   {efface la fin de ligne, utilise le caractère spécial de fond ,
    sinon ' '}

Procedure GotoXY(X,Y : byte);
   {déplace le curseur en colonne X, ligne Y relativement dans la
    fenêtre}

Procedure HighVideo;
   {sélectionne la haute intensité}

Procedure LowVideo;
   {sélectionne la basse intensité}

Procedure NormVideo;
   {sélectionne l'intensité normale}

Procedure TextBackground(Couleur : byte);
   {initialise la couleur de fond pour les écritures sur l'écran}
```

```
Procedure TextColor(Couleur : byte),
   {initialise la couleur des caractères pour les écritures sur
    l'écran}

Procedure TextMode(Mode : Word);
   {permet de passer en mode texte}

Function WhereX : byte;
   {renvoie la coordonnée relative de la colonne du curseur dans la
    fenêtre}

Function WhereY : byte;
   {renvoie la coordonnée relative de ligne du curseur dans la
    fenêtre}

Procedure Window (X1, Y1, X2, Y2 : byte);
   {initialise les coordonnées de la fenêtre actuelle}
```

Routines appartenant également à l'unité TPCRT

En général, les routines qui suivent sont fonctionnellement identiques aux routines de même nom dans TPCRT. Les seuls points importants qui méritent d'être notés sont :

1. *PlaceEnXYAbs* ne fait rien si l'écran actif est un écran virtuel et si *X* et *Y* sont en dehors de la fenêtre active.
2. *DerouleFenHaut* et *DerouleFenBas*, concernent toujours la page active de l'écran physique, même si l'écran actuel est un écran virtuel.

```
Procedure CadreFen(colG, ligneSup, colR, ligneInf, AttrC, AttrE : byte;
                   Entetes : string);
   {dessine un cadre autour d'une fenêtre}

Procedure PlaceEnXYAbs(X,Y : byte);
   {déplace le curseur sur la colonne X et la ligne Y en  mode
    absolu, ne fait rien si l'écran actif est un écran virtuel et si X
    et Y sont en dehors de la fenêtre active}

Procedure DerouleFenBas(X1, Y1, X2, Y2 ,Lignes : byte);
   {déroule la fenêtre sélectionnée vers le bas du nombre de lignes
    spécifiées, concerne toujours la page active de l'écran physique}

Procedure DerouleFenHaut(X1, Y1, X2, Y2, lignes : byte);
   {déroule vers le haut la fenêtre sélectionnée du nombre de lignes
    spécifiées, concerne toujours la page active de l'écran physique}

Function CoordXY : Word;
   {renvoie les coordonnées ligne et colonne absolues du curseur, le
    bit de poids fort contient la ligne actuelle (Y), le bit de poids
    faible contient la colonne actuelle (X)}

Function CoordXAbs : byte;
   {renvoie la coordonnée de colonne absolue du curseur }

Function CoordYAbs : byte;
   {renvoie la coordonnée ligne absolue du curseur}
```

TPECRAN : Déclarations

Constantes

```
Black : byte = 0;
Blue : byte = 1;
Green : byte = 2;
Cyan : byte = 3;
Red : byte = 4;
Magenta : byte = 5;
Brown : byte = 6;
LightGray : byte = 7;
DarkGray : byte = 8;
LightBlue : byte = 9;
LightGreen : byte = 10;
LightCyan : byte = 11;
LightRed : byte = 12;
LightMagenta : byte = 13;
Yellow : byte = 14;
White : byte = 15;
```

En redéfinissant ces identificateurs (déjà déclarés dans TPCRT) comme des constantes typées, TPECRAN pourra réadapter les couleurs lors des affichages sur des écrans monochromes ou noir et blanc mixtes.

```
EcransMax = 10;
```

Nombre maximum d'écrans virtuels que TPECRAN peut gérer.

```
Turbo3StyleCouleurs : boolean = False;
```

Si = "True", *HighVideo/NormVideo/LowVideo* se comporteront comme sous la version 3.x de Turbo (*NormVideo* sélectionne le jaune sur fond noir, etc).

Types

```
CoordFen = Record
                XB, YB, XH, YH : byte;
           End;
```

Enregistrement utilisé pour sauvegarder/restaurer les coordonnées d'une fenêtre.

```
EnrEcran =
Record
   Alloc : boolean ;        {est-il alloué ?}
   Taille : Word;           {taille en mots}
   ECols,                   {nombre de colonnes}
   ELignes,                 {nombre de lignes}
```

```
    {coordonnées fenêtres}
    XBas,                    {0..79}
    YBas,                    {0..24/42/49}
    XHaut,                   {1..80}
    YHaut,                   {1..25/43/50}
    Col,                     {coordonnées curseur}
    Ligne : Byte;
    BPtr : Pointer;          {pointeur sur le buffer écran}
  End;
```

Cet enregistrement décrit un écran donné virtuel ou physique. *Alloc* indique si un buffer a été ou non alloué dynamiquement. *Taille* contient la taille totale du buffer écran en mots. *ECols* et *ELignes* contiennent la hauteur et la largeur de l'écran. *XBas, YBas, XHaut* et *YHaut* contiennent les coordonnées *XY* de la fenêtre actuelle sur l'écran. *Col* et *Ligne* contiennent la position actuelle (absolue) du curseur. *BPtr* pointe sur le buffer de l'écran.

Variables

```
PageActive : Byte;
```

Page vidéo actuellement affichée.

```
EcranActuel : Byte;
```

Numéro de l'écran actif (écran physique = 0)

```
SortieEcran : boolean ;
```

"True" si la sortie est dirigée sur l'écran physique (utilisé de façon interne par TPECRAN).

```
Ecrans : Array[0..EcransMax] of EnrEcran;
```

Tableau des descripteurs d'écran virtuel.

```
PageVisible : Byte Absolute PageActuelle;
```

Page affichée visible sur l'écran (semblable à PageActuelle dans TPCRT).

AlloueEcran

Déclaration

```
Function AlloueEcran(Identificateur, Cols, Lignes : Byte) : boolean ;
```

Objet

Définit un écran virtuel et lui alloue un buffer.

Commentaires

Cette routine définit un écran virtuel de dimensions spécifiées et alloue un buffer. Sauf dans les cas où vous gérez vous-même vos propres buffers d'écrans (en utilisant *DefinitEcran*) *AlloueEcran* devra être appelée avant n'importe quelle autre opération sur un écran virtuel.

Si un buffer a déjà été alloué pour le même écran, il sera libéré avant que le nouveau ne soit alloué. *Identificateur* doit appartenir à l'intervalle *1..EcransMax*.

Exemples

```
Succes := AlloueEcran(0, ColsActuelles, LignesActuelles);
```

Succes sera initialisé à "False". Aucun buffer ne doit être alloué pour l'écran physique de numéro zéro.

```
Succes := AlloueEcran(1, ColsActuelles, LignesActuelles);
```

Définit l'écran numéro 1 avec les mêmes dimensions que l'écran actif. *Succes* sera initialisé à "False" seulement si un buffer écran de la taille désirée ne peut pas être alloué.

Voir aussi

DeallouéEcran (TPECRAN) DefinitEcran (TPECRAN)
CreeEcranV (TPFENETR).

AssignCrt

Déclaration

```
Procedure AssignCrt(var F : Text);
```

Objet

Utilisé pour prendre le contrôle sur la sortie standard en modifiant le bloc d'interface du fichier (FIB : File Interface Block).

Commentaires

Cette routine est destinée la plupart du temps aux utilisations internes. Comme les routines *AssignCrt* de TPCRT et de l'unité standard CRT, elle est utilisée pour réinitialiser les pointeurs de sortie sur le propre module d'entrée/sortie de TPECRAN. Contrairement à ces deux autres routines, elle n'est pas utilisée pour prendre le contrôle sur l'entrée standard (en modifiant l'entrée).

Voir aussi

InitVideo (TPECRAN).

ColsActuelles

Déclaration

```
Function ColsActuelles : Byte;
```

Objet

Retourne le nombre de colonnes dans la fenêtre active.

Commentaires

Cette fonction renvoie la largeur de la fenêtre active.

Voir aussi

LignesActuelles (TPECRAN)

ColsEcranV (TPFENETR).

CopieEcran

Déclaration

```
Procedure CopieEcran(Repere : Byte);
```

Objet

Copie le buffer écran tout entier sur l'écran actif.

Commentaires

CopieEcran ressemble à *CopieFen* mais ne se soucie pas des coordonnées de la fenêtre. Elle tente de recopier tout le buffer écran spécifié dans le buffer de l'écran actif.

Si l'écran actif n'a pas les mêmes dimensions (le même nombre de lignes et de colonnes) que l'écran spécifié par *Repere* (écran source), *CopieEcran* essaiera de restructurer l'écran actif. Cette opération aboutira seulement si le buffer de l'écran actif est aussi grand ou plus grand que le buffer de l'écran source.

Supposez, par exemple, que l'écran actif soit un écran de dix colonnes et quatre lignes, et que l'écran source ait six colonnes et six lignes. Un buffer écran a une taille de 2 * (lignes * colonnes), si bien que le buffer de l'écran actif pourra contenir 80 octets (2 *(4 * 10)) et le buffer de l'écran source contient seulement 72 octets (2 * (6 * 6)). Puisque l'écran actif est plus grand que l'écran source, *CopieEcran* changera simplement les dimensions de l'écran actif en six colonnes par six lignes (mais pas la taille du buffer lui-même) puis recopiera les données.

Si deux écrans ont les mêmes dimensions, aucun recadrage n'est nécessaire et le contenu de l'écran source est immédiatemment recopié sur l'écran actif.

Outre la recopie de l'écran lui-même, *CopieEcran* positionnera également le curseur à la coordonnée relative qu'il avait dans la fenêtre inactive.

Exemple

```
var
   I : Word;
begin
   ClrScr;
   for I := 1 to 24 do begin
      EcritCentre('Bonjour à tous !');
      WriteLn;
   end;
   Write('Pressez une touche pour continuer..');
   I := LitMotTouche;
   Write(^M);
   ClrEol;
   if not AlloueEcran(1, ColsActuelles, LignesActuelles) then Halt;
   SelectEcran(1);
   CopieEcran(0);  {copie le contenu de l'écran 0 dans l'écran 1}
   SelectEcran(0);
   ClrScr;
   GotoXY(1, 25);
   Write('Pressez une touche pour restaurer l''écran...');
   I := LitMotTouche;
   CopieEcran(1);   {copie le contenu de l'écran 1 dans l'écran 0}
end.
```

Voir aussi

CopieFen (TPECRAN) DeplaceEcrVDansFen (TPFENETR).

CopieFen

Déclaration

```
Procédure CopieFen(Repere : Word);
```

Objet

Recopie une fenêtre d'un écran spécifié sur un écran actif.

Commentaires

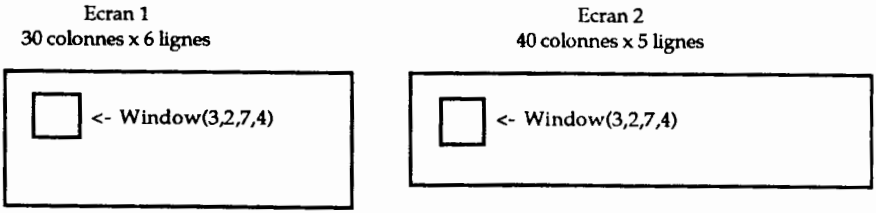
Comme *CopieEcran*, cette routine recopie les données d'un autre buffer écran sur l'écran actif. Néanmoins, *CopieFen* ne se soucie pas des coordonnées de la fenêtre. La meilleure façon de comprendre ce qu'elle fait est de "voir" ce qu'elle fait. Pour cela, examinons les cas suivants :

Cas 1 : les deux écrans ont les mêmes dimensions et la fenêtre a la taille de l'écran tout entier, dans ce cas *CopieFen(N)* se comportera exactement comme *CopieEcran(N)*.

Cas 2 : l'écran 1, l'écran actif, a été défini avec un *maximum* de trente colonnes et de six lignes ; l'écran 2 a été défini avec quarante colonnes et cinq lignes. Pour l'écran 1, les coordonnées de la fenêtre actuelle seront initialisées avec *Window(3,2,7,4)* ; (fenêtre marquée 'A'). Supposons maintenant que nous voulions remplir cette petite fenêtre avec le texte de l'écran 2 ; il suffit d'écrire :

```
CopieFen(2);
```

En résultat, le contenu de la petite fenêtre marquée B sera recopié dans la fenêtre marquée A, (voir illustrations ci-dessous).



Cas 3 : identique au cas 2 sauf que les coordonnées de la fenêtre de l'écran 1 seront initialisées avec *Window(1,1,30,6)* ; (écran total). Dans ce cas, puisque l'écran 2 est plus large que l'écran 1, seules les trente premières colonnes (et les cinq lignes) de l'écran 2 seront déplacées, remplissant toutes les lignes du bas de l'écran 1.

Outre la recopie de la fenêtre elle-même, *CopieFen* essaie également de positionner le curseur au même emplacement relatif que dans la fenêtre inactive.

Exemple

```
var
   I : Word;
begin
   ClrScr;
   if not AlloueEcran(1, 30, 6) then Halt;
   if not AlloueEcran(2, 40, 5) then Halt;
   SelectEcran(2);
   Window(3, 2, 7, 4);
   RemplitFen('2');
   SelectEcran(1);
   Window(3, 2, 7, 4);
   RemplitFen('1');
   SelectEcran(0);          {copie l'écran sur l'écran physique}
   GotoXY(1, 25);
   Write('Pressez une touche pour voir le contenu de l''écran 1');
   I := LitMotTouche;
   CopieFen(1);
   GotoXY(1, 25);
   Write(^M'Pressez une touche pour voir le contenu de l''écran 2');
   I := LitMotTouche;
   CopieFen(2);
   GotoXY(1, 25);
end.
```

Cet exemple est semblable au cas 2 ci-dessus mais les données sont recopiées sur l'écran physique pour voir les résultats.

Voir aussi

CopieEcran (TPECRAN).

CouleurCouleurs

Déclaration

```
Procedure CouleurCouleurs;
```

Objet

Active l'affichage des couleurs.

Commentaires

Destinée principalement aux usages internes, cette routine sélectionne l'affichage des couleurs par défaut au lieu de l'affichage des couleurs monochromes décrit rapidement dans les explications de *CouleursMono*. Puisque TPECRAN sélectionne automatiquement des couleurs appropriées en fonction du mode vidéo actuel, cette routine ne sera probablement appelée que pour des texsts.

Voir aussi

CouleursMono (TPECRAN).

CouleursMono

Déclaration

```
Procedure CouleursMono;
```

Objet

Passe de l'affichage des couleurs à l'affichage monochrome/noir et blanc.

Commentaires

Conçue principalement pour les usages internes, cette routine sélectionne l'adaptation des couleurs en monochromes de telle façon que les couleurs sélectionnées pour un moniteur couleur resteront acceptables sur un moniteur monochrome ou un moniteur mixte noir et blanc. Par exemple, lorsque *CouleursMono* a été appelée,

```
TextColor(Yellow);
TextBackground(Black);
```

auront les mêmes effets que

```
TextColor(White);
TextBackground(Black);
```

Puisque TPECRAN sélectionne automatiquement les couleurs appropriées en fonction du mode vidéo actuel, cette routine sera appelée seulement pendant les tests de votre programme afin que vous puissiez voir les effets de l'adaptation des couleurs.

Voir aussi

CouleurCouleurs (TPECRAN).

DealloueEcran

Déclaration

```
Procedure DealloueEcran(Identificateur : Byte);
```

Objet

Libère le buffer écran alloué par *AlloueEcran*.

Commentaires

Cette routine ne pourra libérer un buffer qui n'a pas été alloué par *AlloueEcran*. S'il n'y pas de buffer alloué, elle ne fera rien. *Identificateur* doit être dans l'intervalle *1..EcranMax*.

Exemples

```
DealloueEcran(0);
```

N'a aucun effet puisque 0 est le repère de l'écran physique.

```
DealloueEcran(1);
```

Libère le buffer pour la fenêtre numéro 1.

Voir aussi

AlloueEcran (TPECRAN) LibereEcranV (TPFENETR).

DefinitEcran

Déclaration

```
Procedure DefinitEcran(AdrBuff : Pointer;
                       Identificateur, cols, lignes : Byte);
```

Objet

Définit un écran virtuel.

Commentaires

Vous n'aurez pas à utiliser cette routine de bas niveau sauf si vous voulez gérer vous-même vos buffers d'écran. Elle diffère légèrement de la routine *AlloueEcran* car elle suppose que vous avez déjà alloué un buffer (pointé par *AdrBuf*) et que vous l'avez initialisé. *Identificateur* est le numéro de l'écran défini et doit appartenir à l'intervalle *1..EcranMax*. *Cols* et *lignes* spécifient les dimensions de l'écran.

Exemple

```
DefinitEcran(MonPointeurBuffer^,1,ColsActuelles,LignesActuelles);
```

Incorrect ! Le '^' n'est pas nécessaire.

```
DefinitEcran(MonPointeurBuffer,1,ColsActuelles,LignesActuelles);
```

Définit l'écran 1 avec les mêmes dimensions que l'écran actif. *MonPointeurBuffer* pointe sur le buffer écran.

Voir aussi

AlloueEcran (TPECRAN) CreeEcranV (TPFENETR).

DeplaceFenHoriz

Déclaration

```
Procedure DeplaceFenHoriz(Identificateur, Cols : Byte;
                          DepDroite : Boolean);
```

Objet

Déplace la fenêtre active de *Cols* colonnes sur la gauche/droite en remplissant l'arrière plan avec le texte depuis l'écran spécifié.

Commentaires

Cette routine déplace le contenu de la fenêtre active à gauche (*DepDroite* = "False") ou à droite (*DepDroite* = "True") du nombre de colonnes spécifié (*Cols*). Comme *DeplaceFenVerti*, elle utilise le texte d'un autre écran (spécifié par *Identificateur*) pour le remplissage au lieu d'afficher des colonnes blanches. L'écran utilisé en arrière-plan doit avoir les mêmes dimensions (nombre de lignes et de colonnes) que l'écran actif sinon la routine ne fera rien.

Supposons que la fenêtre pour l'écran actif (0) soit une fenêtre de 80 colonnes par 25 lignes et que l'écran soit vide sauf la première colonne qui est remplie de haut en bas par la lettre ('O') et que l'écran 2 soit vide sauf la dernière colonne qui est remplie de la même façon de haut en bas avec la lettre '2'. Si nous exécutons l'instruction *DeplaceFenHoriz (2, 1, False);* la colonne remplie avec '0' sera déplacée en dehors de l'écran 0 (de façon permanente) et la colonne remplie avec '2' apparaîtra à droite de celui-ci (voir l'exemple ci-dessous).

Exemple

```
var I : Word; Ch : Char;
begin
   ClrScr;
   for I := 1 to 24 do Writeln('0');
   if not AlloueEcran(2, 80, 25) then Halt;
   SelectEcran(2);
   ClrScr;
   for I := 1 to 24 do begin
      GotoXY(80, I); Write('2');
   end;
   SelectEcran(0);
   Write('Pressez une touche...');
   Ch := ReadKey;
   DeplaceFenHoriz(2, 1, False);
end.
```

Voir aussi

DeplaceFenVerti (TPCRT) DeplaceFen (TPFENETR).

DeplaceFenVerti

Déclaration

```
Procedure DeplaceFenVerti(Identificateur , Lignes : Byte;
                          DepBas : Boolean) ;
```

Objet

Déplace la fenêtre active de *Lignes* lignes en haut ou en bas en remplissant l'arrière-plan avec un texte depuis l'écran spécifié.

Commentaires

Cette routine déplace le contenu de la fenêtre active vers le haut (*DepBas* = "False") ou vers le bas (*DepBas* = "True") du nombre de lignes spécifié par *Lignes*. Contrairement à *DerouleFenHaut* et *DerouleFenBas*, elle utilise les lignes d'un autre écran (spécifié par *Identificateur*) comme écran de remplissage au lieu de créer des lignes blanches et elle peut être utilisée avec des écrans virtuels. L'écran en arrière-plan doit avoir les mêmes dimensions (nombre de lignes et de colonnes) que l'écran actif sinon la routine ne fera rien.

Supposons que la fenêtre de l'écran actif (0) comporte 80 colonnes par 25 lignes, que l'écran soit vide sauf la première ligne qui contient, 'ligne 1, écran 0' et que l'écran 2 soit vide sauf la dernière ligne qui contient, 'ligne 25, écran 2'. Si nous exécutons l'instruction *DeplaceFenVerti(2,1,False);* alors 'ligne 1, écran 0' sera déplacé hors de l'écran (de façon permanente) et 'ligne 25, écran 2' apparaîtra en bas de l'écran (voir exemple ci-dessous).

Exemple

```
var Ch : Char;
begin
   ClrScr;
   Writeln('Ligne 1, écran 0');
   if not AlloueEcran(2, 80, 25) then Halt;
   SelectEcran(2);
   ClrScr;
   GotoXY(1, 25);
   Write('Ligne 25, écran 2');
   SelectEcran(0);
   Write('Pressez une touche...');
   Ch := ReadKey;
   DeplaceFenVerti(2, 1, False);
end.
```

Voir aussi

DeplaceFenHoriz (TPCRT)
DerouleFenBas (TPCRT)
DerouleFenHaut (TPCRT)
DeplaceFen (TPFENETR).

EcritCentre

Déclaration

```
Procedure EcritCentre(S : string);
```

Objet

Centre une chaîne de caractères sur la ligne actuelle dans la fenêtre active.

Commentaires

Cette routine se superpose à la routine *EcritRapide*. Elle positionne le curseur sur la ligne actuelle de telle façon qu'après avoir appelé *EcritRapide*, *S* apparaîtra centré. Une fois la chaîne écrite, le curseur se place en fin de chaîne.

Exemple

```
SelectEcran(0);
GotoXY(1, 1);
EcritCentre('Cette chaîne est centrée.');
```

Voir aussi

EcritRapide (TPECRAN) CentreVite (TPCRT).

EcritMemVite

Déclaration

```
Procedure EcritMemVite(St : string ; Var Adresse ; Attr : Byte);
```

Objet

Ecrit *St* à *Adresse* avec l'attribut *Attr* (attribut vidéo) sans parasite ni scintillement.

Commentaires

Destinée principalement aux usages internes, (utilisée par *EcritRapide* par exemple), cette routine écrira une chaîne de caractères à une adresse mémoire en utilisant l'attribut *Attr*. Si *Adresse* est une partie de la mémoire vidéo, les parasites et les scintillements seront évités. Cette routine est pratiquement identique à *EcritVite* (TPCRT) qui peut théoriquement écrire à n'importe quelle adresse mémoire mais qui ne calcule pas l'adresse où la chaîne doit être écrite.

Exemple

```
EcritMemVite('Bonjour à tous!',Mem[SegmentVideo:0],TextAttr);
```

Ecrit 'Bonjour à tous !' en haut de l'écran en utilisant l'attribut vidéo par défaut.

Voir aussi

EcritVite (TPCRT).

EcritRapide

Déclaration

```
Procedure EcritRapide(S : string) ;
```

Objet

Ecrit une chaîne de caractères sur l'écran à l'adresse actuelle du curseur avec l'attribut actuel.

Commentaires

EcritRapide allie la simplicité de *Write* du Pascal avec la rapidité de la routine *EcritVite* de TPCRT et (contrairement à *EcritVite*) elle peut être utilisée pour écrire sur des écrans virtuels. Après avoir écrit la chaîne de caractères spécifiée sur l'écran à la position actuelle du curseur, *EcritRapide* déplace le curseur en fin de chaîne exactement comme une instruction *Write* le ferait.

Deux différences cependant. Premièrement, comme *EcritVite*, *EcritRapide* affiche les caractères de contrôle tels que les retours-chariot et les passages à la ligne, si bien que vous ne pouvez pas l'utiliser à la place de *Writeln*. Deuxièmement, si la chaîne de caractères tout entière ne correspond pas à la ligne actuelle, *EcritRapide* tronquera cette chaîne de caractères. Par exemple, si la chaîne a une longueur de 80 caractères, la fenêtre a une largeur de 80 caractères et que le curseur est en colonne 1, le dernier caractère de la chaîne ne sera pas affiché ; finalement si le curseur est en dehors des limites de la fenêtre active, *EcritRapide* ne fera rien.

Exemples

```
EcritRapide('Bonjour !'^M^J);
```

Ce n'est probablement pas ce que vous désiriez. Le ^M(retour chariot) et ^J(passage à la ligne) seront affichés comme une note et un carré avec un cercle à l'intérieur.

```
EcritRapide(Chaine1,Chaine2);
```

Incorrect. Contrairement à *Write*, vous ne pouvez passer qu'un seul paramètre à *EcritRapide* à la fois.

```
EcritRapide(Chaine1+Chaine2);
```

Correct.

Voir aussi

EcritVite (TPCRT)
EcritViteEtCoupe (TPFENETR)
EcritMemVite (TPECRAN)
EcritEcranV (TPFENETR).

EnregistreCoordsFen

Déclaration

```
procedure EnregistreCoordsFen(Var CF : CoordFen);
```

Objet

Enregistre les coordonnées de la fenêtre actuelle.

Commentaires

Cette routine permet de sauvegarder les coordonnées de la fenêtre active. Elles peuvent être restaurées ultérieurement en utilisant *RestaureCoordsFen*. Remarquez que le même écran doit être actif lorsque vous restaurez les coordonnées de la fenêtre qui était active lorsque vous les avez sauvegardées.

Exemple

```
var
   CF : CoordFen;
begin
   EnregistreCoordsFen(CF);
   Window(1, 1, 80, 10);
   {...Suite...}
   RestaureCoordsFen(CF);
end.
```

Voir aussi

RestaureCoordsFen (TPECRAN) EnregistreCoordsFen (TPCRT).

InitCarFond

Déclaration

```
Procedure InitCarFond(Car : Char);
```

Objet

Initialise le caractère de fond devant être utilisé par *ClrScr* et *ClrEol*.

Commentaires

Normalement, *ClrScr* et *ClrEol* utilisent un espace (' ') pour remplir l'écran. Cette routine vous permet de sélectionner un autre caractère pour le remplissage.

Exemple

```
InitCarFond(#177);
ClrScr;
```

Remplit l'écran actif avec le caractère #177.

Voir aussi

CaracTexte (TPCRT)

InitPageActive

Déclaration

```
Procedure InitPageActive(NumPage : Byte) ;
```

Objet

Initialise la page vidéo qui sera écrite sur l'écran.

Commentaires

Cela est expliqué dans les généralités de ce chapitre, la page active est la page vidéo actuelle où s'effectuent les écritures, par opposition à la page visible qui est celle actuellement affichée. *InitPageActive* n'a aucun effet si un adaptateur couleur n'est pas utilisé, puisque les adaptateurs monochromes ont seulement une page vidéo.

InitPageActive ne sélectionne pas l'écran physique s'il n'est pas déjà sélectionné. Pour cela, vous devez utiliser '*SelectEcran (0);*'.

Exemple

```
InitPageActive(1);
ClrScr;
writeln('Bonjour!');
InitPageVisible(1);
```

Fait de la page 1 la page active, efface l'écran, écrit une chaîne de caractères puis fait de la page 1 la page visible pour que vous puissiez voir ce qui a été écrit.

Voir aussi

InitPageVisible (TPCRT).

InitVideo

Déclaration

```
Procedure InitVideo;
```

Objet

Initialise les variables et les structures de données de TPECRAN.

Commentaires

Puisque TPECRAN est initialisée automatiquement, cette routine sera rarement appelée ; le seul cas où vous pourriez l'utiliser serait pour réactiver les routines vidéo de TPECRAN après un appel à *LibereVideo* qui les désactive. L'appel à *InitVideo* n'a aucun effet si l'initialisation a déjà été faite.

Voir aussi

LibereVideo (TPECRAN)

LibereVideo

Déclaration

```
Procedure LibereVideo ;
```

Objet

Déroute toutes les sorties vers TPCRT.

Commentaires

Cette routine désactive la plupart des routines d'affichage de TPECRAN et par ce moyen déroute les sorties écran vers TPCRT. L'utilisation la plus typique est d'éviter l'écriture des messages affichés par un module de sortie sur un écran virtuel plutôt que sur l'écran physique. Bien que TPECRAN appelle *LibereVideo* automatiquement lorsqu'un programme se termine, il faut l'appeler pour permettre à tous les messages générés par un module de sortie de Turbo de s'achever, cet appel ne sera pas effectif tant que vos modules de sortie ne seront pas exécutés.

Exemple

```
var
   SauveProcSortie : Pointer;
...
{$F+}
Procedure ProcSortie;
begin
   ExitProc := SauveProcSortie;
   LibereVideo;
   writeln('Erreur ', ExitCode, ' à l''adresse ',
            MotHex(Seg(ErrorAddr^)), ':', MotHex(Ofs(ErrorAddr^)));
end;
{$F-}
...
SauveProcSortie := ExitProc;
ExitProc := @ProcSortie;
```

Voir aussi

InitVideo (TPECRAN).

LignesActuelles

Déclaration

```
Function LignesActuelles : Byte;
```

Objet

Renvoie le nombre de lignes de la fenêtre active.

Commentaires

Cette fonction renvoie la hauteur de la fenêtre active.

Voir aussi

ColsActuelles (TPECRAN) LignesEcranV (TPFENETR).

LitLigneVite

Déclaration

```
Function LitLigneVite(Identificateur, Ligne : Byte) : string ;
```

Objet

Lit une ligne complète de texte à partir de n'importe quel écran et la renvoie en résultat.

Commentaires

Cette routine est semblable à la routine *LitVite* de TPCRT mais elle est faite pour lire des lignes complètes et contrairement à *LitVite*, elle fonctionne avec des écrans virtuels aussi bien qu'avec un écran physique. Etant donné un écran (*identificateur*) et un numéro de ligne (*Ligne*), elle renverra toute la ligne en supprimant les espaces de fin.

Exemple

```
SelectEcran(0);
WriteLn('Bonjour à tous !');
WriteLn(LitLigneVite(0,1));
```

Affichera 'Bonjour à tous !' une deuxième fois.

Voir aussi

LitVite (TPCRT) LireEcranV (TPFENETR).

ReinitEcran

Déclaration

```
procedure ReinitEcran;
```

Objet

Réinitialise l'écran.

Commentaires

Cette routine de TPECRAN est équivalente à *ReinitVideo* de TPCRT. *ReinitEcran* réinitialise toutes les variables globales concernant le mode vidéo, la taille de l'écran et le type de l'écran. Elle sélectionne automatiquement l'écran physique comme écran actif, la page visualisée comme page active et le plein écran comme fenêtre active.

ReinitEcran doit être appelée après chaque changement de mode avant de décider d'écrire sur l'écran physique. Si vous utilisez *Exec* (DOS) ou *ExecDos* (TPDOS) pour exécuter un autre programme, souvenez-vous que le programme que vous exécutez peut modifier le mode vidéo. Cela signifie que *ReinitEcran* devra probablement être appelée après chaque appel à *Exec* ou *ExecDos*. De la même façon, si vous utilisez TPECRAN dans des programmes résidents, vous aurez normalement besoin d'appeler *ReinitEcran* chaque fois que votre routine "popup" sera appelée.

Exemple

```
if ModeActuel <> 7 then begin
   TextMode(1);
   ReinitEcran;
   Writeln('Mode 40 colonnes. Pressez une touche...');
   Ch := ReadKey;
   TextMode(3);
   ReinitEcran;
   writeln('Mode 80 colonnes.');
end
else
   Writeln('Carte couleur nécessaire.');
```

Remarquez que *ReinitEcran* est appelée après chaque appel à *TextMode*.

Voir aussi

ReinitVideo (TPCRT) SelectEcran (TPECRAN).

RemplitCaracAttr

Déclaration

```
Procedure RemplitCaracAttr(var Dest; Nombre : Word;
                           Car : Char; Attr : Byte);
```

Objet

Remplit *Dest* avec un nombre de couples caractère/attribut égal à *Nombre*.

Commentaires

Destiné principalement aux usages internes (avec *RemplitFen* et *RemplitCaracVideo*), cette routine remplit toute zone contiguë de mémoire avec le caractère et l'attribut vidéo spécifiés. Si *Dest* est une zone de la mémoire vidéo, les parasites et les scintillements sont supprimés.

Exemple

```
RemplitCaracAttr(Mem[SegmentVideo:0], LignesActuelles * ColsActuelles,
                 '$', TextAttr);
```

Remplit entièrement l'écran (physique) avec des signes dollar ($) en utilisant l'attribut vidéo par défaut.

Voir aussi

RemplitCaracVideo (TPECRAN) RemplitFen (TPECRAN).

RemplitCaracVideo

Déclaration

```
Procedure RemplitCaracVideo(L, C : Byte; Nombre : Word; Car : Char;
                            Attr : Byte);
```

Objet

Remplit un nombre de couples caractère/attribut égal à *Nombre* en ligne *L*, colonne *C*.

Commentaires

Destinée principalement aux usages internes (avec *ClrEol* de TPECRAN) cette routine remplira toute zone contiguë de l'écran avec le caractère et l'attribut vidéo spécifié. *Remarques* : bien que la routine vérifie si le point de départ est à l'intérieur de la fenêtre active, elle ne vérifie pas si le nombre de couples caractère/attribut tiendra dans la fenêtre.

Exemple

```
SelectEcran(0);
RemplitCaracVideo(1, 1, ColsActuelles, '*', TextAttr);
```

Remplit la ligne supérieure de l'écran avec des astérisques en utilisant l'attribut vidéo par défaut. Exemple équivalent à celui donné pour *RemplitFen*.

Voir aussi

RemplitFen (TPECRAN)

RemplitVite (TPCRT).

RemplitFen

Déclaration

```
procedure RemplitFen(Car : Char );
```

Objet

Remplit la fenêtre active avec le caractère spécifié.

Commentaires

Destinée principalement aux usages internes (avec *ClrScr* de TPECRAN), cette routine remplira la fenêtre toute entière avec le caractère spécifié. Elle utilise l'attribut vidéo actuel (*TextAttr*) qui peut être modifié directement (*'TextAttr := $07'*) ou indirectement (en utilisant *TextColor* et *TextBackground*).

Exemple

```
SelectEcran(0);
Window(1,1,80,1);
RemplitFen('*');
```

Crée une fenêtre d'une ligne en haut de l'écran physique et la remplit avec des astérisques.

Voir aussi

RemplitCaracVideo (TPECRAN) EffaceEcranV (TPFENETR).

RestaureCoordsFen

Déclaration

```
Procedure RestaureCoordsFen(CF : CoordFen) ;
```

Objet

Restaure les coordonnées d'une fenêtre sauvegardées précédemment.

Commentaires

Cette fenêtre vous permet de restaurer les coordonnées d'une fenêtre sauvegardées précédemment en utilisant *EnregistreCoordsFen*. Remarquez que le même écran devra être actif quand vous restaurez les coordonnées de la fenêtre qui était active lorsque vous les avez sauvegardées.

Voir l'entrée de *EnregistreCoordsFen* pour exemple.

Voir aussi

EffaceFen (TPFENETR) EnregistreCoordsFen (TPECRAN)
RestaureCoordsFen (TPCRT)

SelectEcran

Déclaration

```
Procedure SelectEcran(Repere : Byte);
```

Objet

Déroute les écritures écran sur la fenêtre spécifiée.

Commentaires

Cette routine vous permet de spécifier où les écritures écran seront dirigées. Si *Repere* = 0, elle déroute les sorties sur l'écran physique sinon sur un écran virtuel. Pour afficher un écran virtuel (appelons-le *NumEcran*), vous devez (1) faire de l'écran physique l'écran actif en utilisant '*SelectEcran(0);*' puis (2) recopier le contenu de l'écran virtuel sur l'écran physique en utilisant '*CopieEcran(NumEcran);*'

Exemple

```
var
   I : Word;
begin
   if not AlloueEcran(1,ColsActuelles,LignesActuelles) then Halt;
   SelectEcran(1); ClrScr;
   for I := 1 to LignesActuelles do begin
      EcritCentre('Bonjour!'); writeln;
   end;
   SelectEcran(0);
   write('Pressez une touche...');
   I := LitMotTouche;
   CopieEcran(1);
end;
```

Voir aussi

CopieEcran (TPECRAN)
ReinitEcran (TPECRAN)
ActiveEcranV (TPFENETR).

3 Saisie de données

Généralités

TPSAISIE, la nouvelle unité de *Turbo Professionnel* 5.0, est un outil très puissant de saisie de données plein écran. Contrairement aux routines de TPEDIT qui sont conçues pour la saisie de nombres et de chaînes uniques, TPSAISIE vous permet de regrouper des champs sur un seul écran. Vous concevez les écrans et TPSAISIE se charge de l'édition, des mouvements du curseur, des contrôles d'intervalle et de la validation des entrées.

TPSAISIE supporte les types de données suivants :

- Caractère : Char, String, Array of Char
- Scalaire : Byte, ShortInt, Word, Integer, LongInt
- Virgule flottante : Real, Single, Double, Extended, Comp, BCD
- Logique : Boolean, Oui/Non
- Date et heure : Date julienne, chaîne date, heure

TPSAISIE offre des interfaces qui vous permettront de l'adapter pour saisir des données de type énuméré ou des types de données définis par l'utilisateur.

La validation des données saisies a lieu en deux étapes. Les masques de saisie permettent un contrôle sommaire pendant l'entrée des champs de données alors que les fonctions internes ou des routines écrites par l'utilisateur sont utilisées pour des contrôles d'intervalle ou d'autres types de contrôle, ou pour des contrôles plus poussés une fois la saisie terminée.

TPSAISIE offre d'autres caractéristiques importantes :

- Une liste de commandes internes pour l'édition et le déplacement dans les champs, ainsi que des interfaces pour ajouter jusqu'à 10 nouvelles commandes.
- Redéfinition du clavier
- Support complet de la souris
- Champs obligatoires et protégés
- Interface pour implémenter des champs calculés

- Champs à choix multiples
- Edition des champs numériques de droite à gauche facultative (mode calculette)
- Déroulement horizontal dans les zones de saisie
- Ecrans de saisie d'au plus 80 colonnes et 127 lignes ; le déroulement de l'écran se fait verticalement pour les zones de saisie plus longues que l'écran physique
- Interface pour implémenter des écrans de saisie de données imbriqués
- Mode lecture seulement pour permettre de "browser" des enregistrements sans les modifier
- Mode mot de passe pour supprimer l'affichage des données saisies
- Séparateur de milliers et symbole de monnaie flottant/fixe pour les champs numériques
- Contrôle programmé de l'affichage de l'aide et des messages d'erreurs
- Interface pour implémenter des tâches en arrière-plan et un système d'aide adaptée au contexte
- Facilité pour modifier l'affichage de la date, de l'heure et des champs numériques pour les calculs avec des unités monétaires différentes
- Nombreuses options pour adapter la modification les champs et la saisie des données
- Adaptation optionnelle des couleurs avec des écrans monochromes.

La nouvelle unité TPDATE complémentaire de TPSAISIE fait également partie de *Turbo Professionnel* 5.0. Elle offre des routines pour calculer des dates juliennes, des chaînes date et des heures. Une date julienne est représentée par un entier long qui contient le nombre de jours écoulés depuis le 1er janvier 1600 et peut contenir n'importe quelle date valide inférieure au 31 décembre 3999. Cette méthode facilite les calculs sur les dates et permet d'enregistrer les dates de manière compacte. Une chaîne date est la représentation d'une date sous forme de chaîne de caractères avec un format : MM/JJ/AA, JJ/MM/AA, MM/JJ/AAAA, etc... Comme pour les dates, les heures sont représentées par des entiers longs qui contiennent le nombre de secondes écoulées depuis minuit, facilitant ainsi les calculs arithmétiques.

TPSAISIE : Saisie de données

Lorsque vous lirez cette partie de la documentation pour la première fois, vous pourrez croire que l'unité TPSAISIE est trop importante et trop complexe à étudier. Mais ne vous découragez pas. Bien qu'elle soit l'unité la plus importante du logiciel, TPSAISIE n'est pas aussi complexe. La taille de la partie interface de l'unité est due principalement à la richesse des possibilités, dont certaines ne seront jamais utilisées mais dont la plupart sont facilement compréhensibles. Le programme exemple SAISIE.PAS montre comment utiliser la plupart d'entre elles.

Pour vous mettre à l'aise avec TPSAISIE, voyons tout d'abord un exemple simple :

```
{$V-} {<-- N'oubliez pas cette directive de compilation avec TpEntree}
Program ExempleSimple;
uses
   TpCrt, TpSaisie;
var
   ESE            : ESenreg;
   CommandeSortie : TypeES;
   MonEnreg : record
                 Nom : string[30];
                 Telephone : string[14];
              end;
begin
   ClrScr;
   FillChar(MonEnreg, SizeOf(MonEnreg), 0);
   InitEnregES(ESE);
   InitAttrMess($0F);
   InitAttrChamp($70);
   InitAttrChaine($70);
   InitAttrCtrl($70);
   AjoutChampString(ESE, 'Nom :', 1, 1, '', 1, 8, 30, 0, Nil,
                    MonEnreg.Nom);
   AjoutChampString(ESE, 'Tél.:', 2, 1, '(99-9) 99-99-99-99', 2, 8,
                    14, 0, Nil, MonEnreg.Telephone);
   CommandeSortie := EcranEdit(ESE, 0, False);
   LibereEcranEdit(ESE);
end.
```

Examinons chaque partie du programme en détails. Dans la partie déclaration, il y a trois variables : *ESE* qui contient les données utilisées par le programme de saisie d'écran pour conserver la trace de tous les champs qui composent l'écran de saisie. *CommandeSortie* contient le code de retour renvoyé par le programme de gestion d'écran une fois que tous les champs ont été modifiés et *MonEnreg* contient les données que vous voulez modifier.

L'instruction *FillChar* au début du programme sert à initialiser les structures de données qui seront modifiées. Cette étape est importante car les éditeurs de champs ignorent si un champ donné contient déjà des données significatives.

InitEnregES doit être appelée avant tout ajout de champs à *ESEnreg* pour vérifier qu'il a été initialisé correctement. Pour éviter que certains oublis n'entraînent des arrêts système, cette routine place dans *ESEnreg* une signature indiquant qu'il a été initialisé. Toute tentative d'ajout de champ dans un enregistrement de saisie d'écran mal initialisé conduira à une erreur fatale signalée par un son bref et à l'arrêt du programme. Cette routine initialise également les coordonnées par défaut de l'écran de saisie de données. Voir la partie "Déroulement des écrans de saisie de données" pour plus de détails.

Les quatre appels de procédures qui suivent définissent les couleurs utilisées pour l'affichage des messages (*InitAttrMess*), des champs inactifs (*InitAttrChamp*), du champ actif (*InitAttrChaine*) et de tout caractère de contrôle du champ en cours de modification (*InitAttrCtrl*). Dans ce cas, les messages sont affichés en blanc sur noir, et les champs actifs et inactifs sont affichés en inversion vidéo. Dans cet exemple, bien que ces couleurs soient initialisées une seule fois, elles peuvent être modifiées librement lors de l'ajout de champs afin de produire des effets spéciaux (vous pourrez par exemple afficher des champs calculés protégés ou obligatoires avec des couleurs spéciales).

Les appels à *AjoutChampString* définissent les deux champs qui composent notre écran de saisie de données. Ces paramètres sont les suivants :

```
procedure AjoutChampString(
   var ESE : ESenreg;              {Ecran de saisie à ajouter}
      Mess : string;               {Message de saisie}
      PLigne, PCol : byte;         {Coordonnées du message}
      Format : string;             {Masque de saisie}
      ELigne, ECol : byte;         {Coordonnées du champ}
      LongEdit : byte;             {Longueur du champ à saisir}
      IndexAide : word;            {Index à passer à la routine d'aide}
      Validation : pointer;
                   {Pointeur vers une routine spéciale de validation}
      var EditChn : string);       {Chaîne à saisir}
```

Entre autres choses, ces paramètres (expliqués en détails plus tard) spécifient le texte et l'emplacement du message associé au champ, le masque de saisie du champ, s'il existe, l'emplacement et la largeur du champ de modification et la variable chaîne à modifier. Vous remarquerez que dans la première zone (*Nom*) il n'y pas de masque de saisie. Ceci indique à TPSAISIE que la chaîne peut contenir n'importe quel caractère. La largeur du champ à modifier (30) est alors utilisée pour définir la longueur

de la chaîne dans ce cas. Dans le seconde zone (*Telephone*), la zone de saisie et la largeur du champ à modifier ont une longueur de 14. Si le masque de saisie était plus long que la largeur du champ à modifier, l'éditeur de champs afficherait seulement la partie de chaîne à modifier à n'importe quel moment et opérerait un déroulement horizontal si nécessaire.

Dans la plupart des cas, les coordonnées écran passées à *AjoutChampString* (et aux autres routines similaires) sont traitées en mode absolu, ce qui signifie que ligne 1, col 1 référencent le coin supérieur gauche de l'écran. Si la routine *InitFenEnt* a été appelée, les coordonnées seront interprétées en mode relatif par rapport à la fenêtre spécifiée. Voir la partie "Déroulement des écrans de saisie de données" pour plus de détails.

AjoutChampString n'est pas la seule routine pour ajouter des rubriques dans un écran de saisie. Des routines similaires existent pour ajouter des rubriques d'autres types : *AjoutChampByte* (Byte), *AjoutChampWord* (Word), *AjoutChampReal* (Real), *AjoutChampDate* (Date julienne) et ainsi de suite...

Complément sur l'ajout de champs

Premièrement, les champs seront ajoutés dans un ordre spécifique en fonction de l'emplacement sur l'écran : de gauche à droite et de haut en bas. Le curseur doit pouvoir se déplacer vers le haut, vers le bas, à gauche ou à droite en fonction de la demande. Imaginez par exemple un écran composé de 9 champs structurés en trois lignes et trois colonnes avec le curseur placé dans le milieu du champ (deuxième ligne, deuxième colonne). Pour placer le curseur dans le champ situé au milieu de la ligne supérieure, vous presserez simplement (↑). Pour placer le curseur au milieu de la ligne inférieure, pressez (↓). Pour placer le curseur sur les colonnes de gauche ou les colonnes les plus à droite, pressez (←) ou (SHIFT)-(TAB), ou (→) ou (TAB) en fonction de la position actuelle du curseur. Contrairement aux autres programmes de saisie plein écran qui n'ont pas généralement la notion d'espace, TPSAISIE fournit une interface logique qui permet à l'utilisateur de se déplacer de champ en champ librement.

Deuxièmement, lorsqu'un champ est ajouté, il reçoit automatiquement un identificateur de champ (un numéro entre 0 et 254). Le premier champ ajouté a le numéro d'identificateur 0 et le second a l'identificateur 1 et ainsi de suite. Cette valeur référence sans ambigüité chaque champ d'un écran de saisie. Le programme de gestion de l'écran peut ainsi indiquer aux routines écrites par l'utilisateur quel est le champ en cours de modification.

Lorsque tous les champs ont été ajoutés, vous pouvez commencer à les modifier en appelant *EcranEdit* qui comporte trois paramètres. Le premier est le nom de l'enregistrement de l'écran de saisie, le second est l'identificateur "ID" du premier champ à modifier. Normalement, vous

devrez spécifier la valeur 0 mais, vous pourrez, si vous le souhaitez, spécifier une autre valeur. Par exemple si le champ supérieur de l'écran est un champ non protégé modifié très rarement, vous pourrez passer un paramètre égal à 1 pour positionner le curseur initialement sur le deuxième champ. Le troisième est utilisé pour sélectionner le mode "lecture seulement" avec lequel l'utilisateur peut se déplacer dans l'écran de saisie de données sans modifier les champs. Cette option est très utile avec des écrans de saisie de données plus grands que l'écran physique mais dans la plupart des cas, ce paramètre vaudra "False".

Comme nous l'avons déjà dit, *EcranEdit* gère automatiquement les demandes de déplacement d'une rubrique à une autre. Quoi qu'il en soit, plusieurs commandes de modification internes référencées globalement comme "commandes de sortie" peuvent indiquer que l'utilisateur désire modifier un enregistrement différent, supprimer toutes les modifications faites sur l'enregistrement actuel ou terminer la session. Vous pouvez aussi définir votre propre commande de sortie. La commande de sortie utilisée est renvoyée par *EcranEdit* comme résultat de la fonction. Dans notre programme exemple, cette commande est enregistrée dans *CommandeSortie* mais n'est jamais suivie d'effet. Dans une application réelle, vous aurez généralement besoin d'écrire des instructions pour traiter tous les cas possibles de commandes de sortie, y compris celles que vous avez définies vous-même. Voir SAISIE.PAS comme exemple de mise en oeuvre.

La dernière ligne du programme exemple, l'appel à *LibereEcranEdit*, n'est pas nécessaire dans ce cas. Elle libère toute la mémoire qui avait été allouée dynamiquement pour les champs ajoutés, puis appelle *InitEnregES* pour réinitialiser *ESenreg*. Comme le programme exemple se termine ici, il n'est pas nécessaire de restituer la mémoire qui a été allouée. Pour des applications importantes, il sera souhaitable de restituer cette mémoire après chaque session de modification. Voir *LibereEcranEdit* pour plus de détails.

Résumé : Les seules procédures que vous devez appeler si vous utilisez TPSAISIE pour la saisie de données sont *InitEnregES* et *EcranEdit*. Dans les autres cas, vous pourrez appeler les quatre routines (*InitEnregES*, *InitAttrChamp*, *InitAttrChaine* et *InitAttrCtrl*) qui définissent les couleurs des rubriques à afficher. Vous pourrez également appeler une ou plusieurs routines telles que *AjoutChampString*, et ajouter un champ à modifier sur l'écran. Vous pourrez également utiliser si vous le souhaitez, des sons de cloches et des sifflements.

Maintenant que vous avez une idée générale sur la mise en oeuvre de TPSAISIE, nous allons poursuivre avec les concepts généraux en commençant par les masques de saisie.

Masques de saisie

Les masques de saisie d'une rubrique indiquent à TPSAISIE non seulement la façon dont les données sont formatées pendant la modification mais également le type des caractères qui peuvent être saisis à certaines positions. Un masque de saisie est composé d'un ou plusieurs caractères appartenant à l'une des trois catégories : caractère de formatage, caractère littéral et caractère semi-littéral.

Les caractères de formatage indiquent les positions dans un champ auxquelles de nouvelles données peuvent être saisies par l'utilisateur, ou les positions d'un texte existant qui peut être modifié. Chaque caractère de formatage est associé avec son propre jeu de caractères et seuls les caractères appartenant à cet ensemble peuvent être saisis à la position correspondante. Par exemple, le masque '999' indique que seuls les chiffres ('0'..'9') et les espaces peuvent être saisis dans un champ donné. Plusieurs caractères de formatage - '!', 'N' et 'L' par exemple - indiquent également que les données saisies seront automatiquement converties en majuscules ou en minuscules.

En général, tout caractère d'un masque de saisie qui n'est pas un caractère de formatage est considéré comme un littéral, ce qui signifie que cet emplacement du champ ne peut être modifié pendant la saisie. Dans le cas des champs de type chaîne, ces caractères littéraux appartiennent de façon permanente à la chaîne en cours de saisie pour être sûr qu'ils s'afficheront comme ils ont été saisis, lorsque vous les afficherez vous-même. Dans les autres cas, les caractères littéraux d'un masque de saisie sont éliminés lors de la conversion d'une chaîne dans le type de données initial et restaurés lors de la conversion en chaîne.

Les caractères semi-littéraux sont semblables aux caractères littéraux, dans la mesure où ils correspondent habituellement aux emplacements d'un champ qui ne peuvent être modifiés pendant la saisie. Les deux caractères monétaires ('c' et 'C'), par exemple, désignent toujours des emplacements d'un champ qui ne peuvent être modifiés. Ce qui apparaîtra sur l'écran et dans la chaîne à modifier est en général un '$' (qui peut être modifié).

Dans deux autres cas, un caractère semi-littéral exige plus qu'une simple substitution. Le signe dollar ($) sera placé à coté d'un champ numérique seulement si le reste du champ est rempli. Si le champ n'est pas rempli, le signe dollar sera placé juste à gauche du chiffre le plus à gauche. De la même manière, le caractère virgule (',') indique une position qui doit être remplie avec une virgule (ou la valeur de *CaracVirgule*) seulement s'il y a un chiffre à placer à sa gauche. Sinon, cette position sera remplie avec un

signe dollar, un signe moins ou un espace. Quoi qu'il en soit, cet emplacement réservé à un caractère semi-littéral ne pourra pas être rempli avec un chiffre entré par l'utilisateur.

Avant que cela ne devienne trop abstrait, regardons les caractères de formatage définis par TPSAISIE, certains d'entre eux seront familiers aux programmeurs dBase :

Caractères	Domaine de définition de TPSAISIE
'X'	#0..#255 (tout caractère)
'!'	#0..#255 (mise en majuscules des caractères)
'L'	#0..#255 (mise en minuscules des caractères)
'n'	'A'..'Z', 'a'..'z', #128..#154, #160..#167, '', '-', '.', ',' (caractères alphabétiques seulement)
'N'	(identique à 'n' mais mise en majuscules)
'l'	(identique à 'n' mais mise en minuscules)
'B'	'V', 'F' (booléen "Vrai"/"Faux", mise en majuscules)
'O'	'O', 'N' (Oui/Non , forçage en majuscules)
'9'	'0'..'9', ' ' (seulement pour les nombres non signés)
'#'	'0'..'9', '', '-', '.' (nombres non signés ou signés)
'1'	Correspond à l'ensemble des caractères définis par l'utilisateur *Ens1Utl* ; mise en majuscules ou en minuscules facultative en fonction de la valeur de *MetEnMajuscule['1']*
'2'..'8'	Correspond à *Ens2Utl..Ens8Utl, MetEnMajuscule['2'..'8']*

Les caractères semi-littéraux

Caractères	Conversion de TPSAISIE
'c'	'$' ou *ChnGaucheMonnaie* (symbole monétaire fixe - gauche)
'C'	'' ou *ChnDroitMonnaie* (symbole monétaire fixe - droit)
'$'	'$' ou *ChnGaucheMonnaie* (symbole monétaire flottant - gauche)
'.'	'.' ou *CaracDecimal* (fixe)
','	',' ou *CaracVirgule* (séparateur)

Note importante : Les caractères semi-littéraux ',' et '$' sont autorisés seulement dans les champs contenant des points décimaux fixes et dans les champs numériques qui ne peuvent pas contenir de point décimal (champ entier, champ réel avec nombre de positions décimales égal à 0, etc...).

De plus, TPSAISIE reconnaît également les caractères semi-littéraux et formats suivants qui sont définis dans l'unité conjointe TPDATE :

Caractères	Domaine de définition ou de conversion de TPSAISIE
'm', 'j', 'a'	'0'..'9' (pour les champs date)
'M', 'J'	Identique
'/'	'/' ou *CaracSlash* (fixe, semi-littéral)
'h', 'm', 's'	'0'..'9' (pour les champs heure)
'H', 'M', 'S'	identique
':'	':' ou *CaracDeuxPoints* (fixe ou semi-littéral)
't'	'p', 'P', 'a', 'A' (dépend de *HeureMajuscule*)
'e'	'm' ou 'M' (dépend de *HeureMajuscule*)

Pour clarifier l'utilisation de ces caractères de formatage et les différencier, voici quelques exemples de masques de saisie et les résultats produits :

Format	Résultat	Remarque
'XXXXXXXXXXXXXXX'	'Bonjour à tous!'	Accepte tous les caractères
'!!!!!!!!!!!!!!!'	'BONJOUR A TOUS!'	Tous les caractères mis en majuscules
'LLLLLLLLLLLLLL'	'bonjour à tous!'	Tous les caractères mis en minuscules
'NNNNNNNNNNNNNN'	'BONJOUR A TOUS '	Caractères alphabétiques seulement, mis en majuscules'
'!!!!!!!!!!!!!!'	'bonjour à tous '	Caractères alphabétiques seulement, forcés en minuscules
'[9999]'	'[123]'	Chiffres et ' ' seulement, littéraux : crochets
'[####]'	'[-123]	Le signe moins est permis
'c999.99'	'$ 12.50'	Signe dollar fixe
'$999.99'	' $12.50'	Signe dollar flottant
'999.99CC'	'123,45 F'	Signe franc (F) fixe
'$###,###.##'	' $1,234.56'	',' séparateur des milliers, '$' flottant
'$###,###.##'	' $12.34'	
'mm/jj/aa'	'01/16/26'	Le mois est précédé d'un zéro
'MM/jj/aa'	' 1/16/26'	Le mois est précédé d'un espace
'mm/jj/aaaa'	'01/16/1926'	
'MM/jj/aaaa'	' 1/16/1926'	
'jj/mm/aaaa'	'16/01/1926'	Format européen
'hh:mm'	'23:30'	Heure en cycle de 24 heures
'hh:mmt'	'11:30p'	heure en format am/pm
'HH:mm te'	' 1:30 pm'	Les heures sont précédés d'un blanc

Remarquez dans ces exemples que les caractères de formatage ne sont pas mélangés dans les sous-champs. Il n'y pas d'exemple de masque tel que '!XXXX' (convertit le premier caractère en majuscules, les autres sont

libres). Bien que ces formats de masques soient légaux, les résultats produits ne sont pas garantis.

Un exemple d'application typique de jeu de caractères défini par l'utilisateur est de limiter la saisie des données dans une rubrique caractère à un intervalle de valeurs non contigües. Par exemple :

```
Ens1Utl := ['G', 'D', 'C'];          {Gauche, Droite, Centre}
MetEnMajuscule['1'] := Majusc;
Ens2Utl := ['m', 'f'];       {masculin ou féminin}
MetEnMajuscule['2'] := Minusc;
Ens3Utl := ['M', 'm', 'F', 'f'];
MetEnMajuscule['3'] := SansChangement;
```

Etant données ces instructions, un masque de saisie de '1' autorisera seulement la saisie de 'G', 'D' et 'C' et convertira les données en majuscules alors qu'un masque de saisie égal à '2' autorisera seulement la saisie de 'm' et 'f' et les données saisies seront converties en minuscules. Un exemple avec '3' est semblable à l'exemple avec '2' mais les caractères ne seront pas modifiés. Bien qu'il y ait d'autres façons de contrôler les données saisies dans de tels champs, c'est en général la méthode la plus facile.

Conventions et monnaies internationales

Comme vous avez pu le remarquer dans certains masques de saisie des exemples ci-dessus, TPSAISIE manipule des données parfois longues afin de pouvoir prendre en compte de nombreuses monnaies internationales et des conventions de date et d'heure. Pour tirer profit de cette caractéristique, souvenez-vous de ce qui suit :

Tout d'abord, il y a 7 caractères de formatage : ':' (séparateur de l'heure), '/' (séparateur de date), '.' (point décimal), ',' (séparateur des milliers), 'c' (symbole de monnaie fixe - gauche), 'C' (symbole de monnaie fixe - droit) et '$' (symbole de monnaie flottant). Deuxièmement, ces caractères correspondent aux six constantes typées globales : *CaracDeuxPoints* (':'), *CaracSlash* ('/'), *CaracDecimal* ('.'), *CaracVirgule* (','), *ChnGaucheMonnaie* ('c' et '$') et *ChnDroitMonnaie* ('C').

Enfin, souvenez-vous que les unités TPDATE et TPSAISIE fournissent des routines qui permettent à vos programmes de s'adapter aux différentes conventions internationales : *DateInternationale* et *HeureInternationale* de TPDATE et *MonnaieInternationale* de TPSAISIE. Ces routines sont capables grâce aux informations *Country* du DOS, non seulement d'initialiser les quatre constantes typées correctement mais aussi de construire un masque de saisie optimisé.

Les descriptions complètes de ces routines sont fournies plus loin dans le chapitre. Puisque la plupart des programmes sont écrits pour être utilisés dans un seul pays, nous suspendrons momentanément les explications sur la construction manuelle des masques de saisie. Comme les masques de saisie de date et heure sont expliqués complètement dans la documentation de TPDATE, nous allons nous concentrer sur les formats monétaires.

Supposons que vous ayez à utiliser un masque de saisie pour un champ qui doit contenir une valeur comprise entre 0.00 et 999,999.99. Voici quelques exemples de configuration de TPSAISIE adaptée à un système monétaire d'un pays donné :

USA

```
ChnGaucheMonnaie  = '$';          {Par défaut}
ChnDroitMonnaie   = '';           {Par défaut}
CaracDecimal      = '.'           {Par défaut}
CaracVirgule      = ','           {Par défaut}
MasqueSaisie1     = '$999,999.99'; {Symbole monétaire flottant}
Resultat          = ' $1,234.56'
MasqueSaisie2     = 'c999,999.99'; {Symbole monétaire fixe}
Resultat          = '$ 12,345.67'
```

Allemagne

```
ChnGaucheMonnaie  = 'DM';         {Par défaut}
ChnDroitMonnaie   = '';           {Par défaut}
CaracDecimal      = ','           {Par défaut}
CaracVirgule      = '.'           {Par défaut}
MasqueSaisie1     = '$$999.999,99'; {Symbole monétaire flottant}
Resultat          = ' DM1.234,56'
MasqueSaisie2     = 'cc999.999,99'; {Symbole monétaire fixe}
Resultat          = 'DM 12.345,67'
```

France

```
ChnGaucheMonnaie  = '';           {Par défaut}
ChnDroitMonnaie   = 'F';          {Par défaut}
CaracDecimal      = ','           {Par défaut}
CaracVirgule      = ' '           {Par défaut}
MasqueSaisie2     = '999 999,99CC'; {Symbole monétaire fixe}
Resultat          = ' 12 345,67 F'
```

Comme vous pouvez le voir, TPSAISIE permet d'adapter des symboles monétaires sur des longueurs variables (1 à 5 caractères) à gauche ou à droite d'un nombre. Pour garantir l'affichage correct des données, vérifiez que le masque de saisie est conforme aux valeurs de *ChnGaucheMonnaie* et *ChnDroitMonnaie*. Si le symbole monétaire est un simple caractère à

gauche, *ChnGaucheMonnaie* devra contenir ce caractère et 'c', sinon '$' apparaîtra en tête du masque de saisie. S'il s'agit d'une chaîne de deux caractères à droite, *ChnDroitMonnaie* contiendra ces deux caractères et 'CC' apparaîtra à la fin du masque de saisie.

Champs et sous-champs

Avant de poursuivre, nous devons définir deux termes clés. Un 'champ' est utilisé pour référencer la partie d'un masque de saisie dont les caractéristiques sont définies par un appel à une routine telle que *AjoutChampString*. Un champ peut posséder son propre message, sa propre couleur, son propre masque de saisie et ainsi de suite.

Un 'sous-champ' est soit un champ tout entier ou un partie de champ qui est délimitée (conformément au masque de saisie) par un caractère littéral ou par l'un des bords du champ. Pour l'instant, dans notre exemple, le champ correspondant à *MonEnreg.Telephone* comporte un masque de saisie qui pourrait être : '(99-9) 99-99-99-99'. Seules les positions contenant '9' pourront être modifiées pendant la saisie. Ce champ comprend donc six sous-champs.

Remarquez cependant que ces généralisations ne s'appliquent pas lorsque vous utilisez les éditeurs numériques et à choix multiples (décrits ci-dessous) qui ne différencient pas les champs et les sous-champs.

Les trois éditeurs de champs

TPSAISIE comporte trois "éditeurs de champs" pour gérer les informations saisies au clavier. Le plus important est l'éditeur chaîne qui peut être utilisé avec la plupart des types de données. L'éditeur numérique est un éditeur type calculette qui permet d'entrer des nombres de droite à gauche et l'éditeur à choix multiples permet à l'utilisateur de sélectionner un élément dans une liste courte. La liste complète des commandes disponibles dans chacun des trois éditeurs sera présentée dans la prochaine section. Voici un bref survol de ces trois éditeurs.

L'éditeur chaîne ressemble à l'éditeur ligne de TPEDIT. L'éditeur chaîne de TPSAISIE comporte un ensemble de commandes plus important afin de permettre des déplacements de champ en champ et la gestion des masques de saisie. Bien qu'il puisse fonctionner avec un large éventail de types de données, y compris tous les types numériques standard, il est orienté vers l'édition des chaînes de caractères plutôt que vers celle des nombres. Il est principalement limité par ses possibilités de gestion de masques de saisie contenant des virgules et des signes dollar flottants. Lors de la modification des champs dont les masques de saisie contiennent l'un de ces caractères de formatage, l'éditeur chaîne doit les supprimer momentanément puis les restaurer lorsque le nombre a été saisi.

L'éditeur numérique est adapté à la saisie et à la modification des nombres. Il fonctionne comme une calculette. Le curseur est toujours positionné sur le coté droit du champ, les chiffres sont toujours saisis à la fin et décalés vers la gauche. Contrairement à l'éditeur chaîne, il affiche correctement les virgules et le signe dollar flottant. Il est spécialement adapté à la saisie de longues suites de nombres.

L'éditeur à choix multiples n'est pas un éditeur au sens normal du terme puisqu'il ne permet pas de saisir ou de modifier du texte. Il permet à l'utilisateur de sélectionner une option dans une courte liste en pressant une touche particulière jusqu'à ce que l'option désirée apparaisse sur l'écran. Il peut être utilisé avec des champs numériques et permet à l'utilisateur d'incrémenter ou de décrémenter une donnée plutôt que de saisir réellement la valeur. Un autre avantage de l'éditeur à choix multiples est la mise à jour instantanée de champs liés entre eux (comme le montre le programme SAISIE.PAS) chaque fois que l'utilisateur incrémente ou décrémente la donnée.

Commandes éditeur

Cette partie décrit brièvement toutes les commandes reconnues par les trois éditeurs de champs. Ces commandes sont classées en trois catégories : commandes de modification, commandes de déplacement dans les champs et commandes de sortie. La présentation est la suivante : nom symbolique de chaque commande (éventuellement), touches affectées et description de la commande. Les abréviations EC (éditeur chaîne), EN (éditeur numérique) et EM (éditeur à choix multiples) sont utilisées pour présenter les différences de fonctionnement de ces trois éditeurs. Si aucune abréviation n'apparaît dans la description, la commande est la même pour les trois éditeurs.

Remarquez que les commandes qui déplacent le curseur de champ en champ, sauteront toujours les champs protégés (expliqués plus tard dans cette partie). Par exemple, la commande pour déplacer le curseur sur le premier champ de l'écran le placera réellement sur le second si le premier champ est protégé. De la même façon, les commandes qui déplacent le curseur de champ en champ peuvent être affectées en mode "enroule". Voir les explications sur *InitModeEnroule* pour plus de détails.

ESgauche	(←),(CTRL)-S	EC : Déplace le curseur à gauche d'un caractère. Si le curseur est au début d'un sous-champ, il est placé à la fin du sous-champ précédent. EN, EM : Place le curseur sur le champ précédent.
ESdroit	(→), (CTRL)-D	EC : Déplace le curseur à droite d'un caractère. Si le curseur est en fin d'un sous-champ, il est placé au début du sous-champ suivant. EN, EM : Place le curseur sur le champ suivant
ESmotGauche	(CTRL)-(←), (CTRL)-A	EC : Déplace le curseur à gauche d'un mot. Si le curseur est au début du premier mot dans un champ, il est placé au début du dernier mot dans le champ précédent. Un "mot" est n'importe quelle suite de caractères non blancs suivie par un ou plusieurs blancs. EN, EM : Déplace le curseur sur le dernier mot du champ précédent.
ESmotDroit	(CTRL)-(→), (CTRL)-F	EC : Déplace le curseur à droite d'un mot. Si le curseur est en fin de champ, il est placé au début du premier mot dans le champ suivant. EN, EM : Déplace le curseur sur le premier mot du champ suivant.
EShome	(HOME), (CTRL)-Q-S	EC : Déplace le curseur au début du champ
ESend	(END), (CTRL)-Q-D	EC : Déplace le curseur en fin du champ
ESdel	(DEL), (CTRL)-G	EC, EN : Supprime le caractère sous le curseur.

ESback	(BACKSPACE), (CTRL)-H, (CTRL)-(BACKSPACE)	EC : Supprime le caractère à gauche du curseur EN : Supprime le caractère sous le curseur
ESdelLigne	(CTRL)-Y	EN, EC : Efface tout le champ
ESdelFinLigne	(CTRL)-(END), (CTRL)-Q-Y	EC : Supprime les caractères depuis le curseur jusqu'à la fin du sous-champ.
ESdelDeb	(CTRL)-(HOME)	EC : Supprime des caractères du début du sous-champ jusqu'au curseur
ESdelMot	(CTRL)-T	EC : Supprime le mot à droite du curseur
ESins	(INS)	EN, EC : Bascule le mode insertion On ou Off. Un curseur plat indique le mot insertion, un curseur fin indique le mode recouvrement.
ESCaracCtrl	(CTRL)-P	EC : Insère des caractères de contrôle. Par exemple, pour insérer un ^G, vous presserez (CTRL)-P-(CTRL)-G. Cette commande est disponible seulement si le masque de saisie le permet et si l'attribut vidéo des caractères de contrôle est différent de l'attribut vidéo des caractères normaux. Si *SaisieEgaleCtrl* vaut "True" (valeur par défaut), les caractères de contrôle sont affichés comme des caractères alphabétiques majuscules (^G est affiché comme 'G' par exemple) avec l'attribut spécifié. Si *SaisieEgaleCtrl* vaut "False", les caractères sont affichés normalement (^O comme un flocon de neige par exemple) avec l'attribut spécifié.
ESrestaure	(CTRL)-R, (CTRL)-Q-L	Restaure le contenu initial du champ et continue l'édition.

ESaide	(F1)	Aide. Si la routine d'aide de l'utilisateur a été définie en positionnant *PrtAideSaisie* à une valeur différente de nil, la frappe de (F1) appellera cette routine avec l'index d'aide pour le champ courant. Si *PrtAideSaisie* vaut nil, la commande est inopérante.
ESchampSuiv	(ENTER), (CTRL)-M	Accepte le contenu du champ et se place sur le champ suivant.
ESchampSup	(↑), (CTRL)-E	Place le curseur sur le dernier champ sous le champ courant, qui commence à la colonne écran inférieure ou égale à la colonne où se trouvait le curseur.
ESchampInf	(↓), (CTRL)-X	Place le curseur sur le dernier champ sous le champ courant qui commence à la colonne écran inférieur ou égale à la colonne où le curseur est positionné.
EStab	(TAB), (CTRL)-I	Place le curseur au début du prochain sous-champ
ESbackTab	(SHIFT)-(TAB)	Place le curseur au début du sous-champ précédent.
ESpremierChamp	(CTRL)-(PGUP), (CTRL)-Q-R	Place le curseur sur le premier champ de l'écran
ESdernierChamp	(CTRL)-(PGDN), (CTRL)-Q-C	Place le curseur sur le dernier champ de l'écran
ESfinSaisie	(CTRL)-(ENTER), (CTRL)-K-D, (CTRL)-K-Q	Fin des modifications. Indique que l'utilisateur désire terminer la session d'édition.

ESenrPreced	(PGUP)	Indique que l'utilisateur désire arrêter de modifier l'enregistrement courant et commencer à modifier le précédent. Sans signification pour les applications type *Database* pour lesquelles un seul écran de saisie est utilisé pour modifier plusieurs enregistrements.
ESenrSuiv	(PGDN)	Indique que l'utilisateur désire arrêter de modifier l'enregistrement courant et commencer à modifier le prochain.
ESquitte	(ESC), (CTRL)-(BREAK)	Arrêt. Indique que l'utilisateur désire abandonner toutes les modifications de l'enregistrement courant. La valeur initiale du champ courant est automatiquement restaurée. Il appartient au programmeur de restaurer les valeurs initiales des autres champs.
ESincChoix	+, (Espace)	EM : Sélection du prochain choix ou incrémentation de la valeur.
ESdecChoix	-	EM : Sélection du champ précédent ou décrémentation de la valeur.
N.A.	.	EC : Si le champ à modifier contient un point décimal fixe, la saisie d'un point placera le curseur sur l'espace immédiatement après le point décimal. Remarquez que cette commande est réellement affectée non pas à '.' (point) mais plutôt à la valeur actuelle de *CaracDecimal* ; aussi si *CaracDecimal* = ',' (comme c'est le cas dans notre pays), cette commande sera affectée à ','.
N.A.	-	EN : Modifie le signe du nombre saisi.

Remarquez que vous pouvez modifier l'affectation de ces touches (sauf les deux dernières). Voir la partie sur "Modification de l'affectation des touches" et l'appendice C pour plus de détails.

Champs obligatoires et protégés

Normalement, tout champ d'un écran de saisie peut être laissé vide. Quoi qu'il en soit, dans certains cas, les champs vides ne sont pas autorisés, aussi vous devrez désigner ces champs comme obligatoires. Pour cela, appelez simplement *InitObligatoire* avant et après l'ajout d'un tel champ :

```
InitObligatoire(On);
{Ajoutez un ou plusieurs champs obligatoires}
InitObligatoire(Off);
```

Lorsqu'un champ est obligatoire, le programme de gestion d'écran vérifie son contenu à chaque fois qu'il est modifié ou lorsque l'utilisateur quitte l'écran de saisie. Si tous les caractères non littéraux d'un champ sont à blanc, le programme de gestion de l'écran génère une erreur. Attention, en cas de remplissage partiel du champ, aucune erreur ne sera générée. Pour être sûr que le champ ne contienne pas de blancs, vous devrez spécifier une routine de contrôle spéciale telle que *BlancsInterdits* lorsque vous ajoutez la rubrique.

De la même façon, dans certains cas vous pourrez protéger certains champs contre les modifications. Vous désirez seulement que le message et la valeur actuelle de ce champ soient affichées mais qu'il ne puisse pas être modifié par l'utilisateur. Dans ces cas, vous appellerez *InitProtection* :

```
InitProtection(On);
{Ajoutez un ou plusieurs champs protégés}
InitProtection(Off);
```

L'exemple de programme SAISIE.PAS illustre l'utilisation des champs protégés.

Champs calculés

Dans certains cas, un champ protégé pourra être le résultat d'un calcul sur un ou plusieurs autres champs. De tels champs sont appelés champs calculés. Bien qu'il ne soit pas possible avec TPSAISIE de manipuler des champs calculés automatiquement, cette unité vous fournit le moyen de le faire facilement.

La solution est la procédure *InitPtrApresEdit* qui vous permet de spécifier un pointeur sur une procédure qui sera appelée après chaque modification avant que le curseur ne soit placé sur un autre champ. En contrôlant la zone *IDCourant* de l'enregistrement de l'écran de saisie passé en paramètre, cette procédure peut déterminer le champ qui vient d'être traité et recalculer la valeur pour tous les champs affectés par la modification. Le schéma de cette procédure pourrait être le suivant :

```
{$F+}
procedure GestChampCalcule(var ESE : ESEnreg);
begin
   {Vérification du champ courant}
   case ESS.IDCourant of
      {Mise à jour de tous les champs affectés
       par la modification du champ courant}
      {Mise à jour de l'écran avec DessineChamp}
   end;
end;
{$F-}
```

Lorsque la valeur d'un champ calculé a été modifiée, vous pouvez mettre à jour l'écran en appelant la routine *DessineChamp* avec l'identificateur du champ qui a été modifié :

```
DessineChamp(ESE, IDChamp);
```

Remarquez que la procédure *CalculChamp* pourrait être appelée chaque fois qu'une valeur d'un champ à choix multiples est modifiée ; cela vous permet de mettre à jour les valeurs des champs qui en dépendent pendant que l'utilisateur fait un choix.

Champs à choix multiples

Comme avec les champs calculés, TPSAISIE vous apporte une certaine aide dans la gestion des champs à choix multiples. Lorsque vous appelez *AjoutChampChoix* pour ajouter un tel champ, vous devez fournir un pointeur sur une routine de conversion des valeurs en chaînes ; cette routine incrémente ou décrémente une valeur lorsque l'utilisateur a fait un autre choix. Le schéma d'une telle routine pourrait être le suivant :

```
{$F+}
procedure IncValeur(var Valeur; ID : Byte; Facteur : Integer; var S :
String);
begin
   case Facteur of
      +1 : {Incrémentation valeur};
      -1 : {Décrémentation valeur};
   end;
   S := {Représentation chaîne de la valeur};
end;
{$F-}
```

La procédure *IncValeur* doit éventuellement transformer le type de *Valeur*. Elle peut utiliser une transformation explicite telle que 'Byte(Valeur)' ou elle peut déclarer une variable absolue du type approprié - 'ValeurByte : Byte absolute Valeur' - et travailler avec cette variable.

Facteur indique si *Valeur* doit être incrémentée ou décrémentée avant de la convertir en chaîne. Si *Facteur* vaut zéro, seule la conversion est nécessaire (*Facteur* aura toujours une valeur comprise entre -1 et 1). *ID* indique quel

champ est associé à *Valeur* dans l'écran de saisie, ce qui facilite l'écriture d'une routine qui traite tous les champs à choix multiples dans un écran de saisie.

Bien que ce schéma soit conçu pour manipuler des types énumérés tels que 'Genre = (Masculin, Féminin, Inconnu)', il peut aussi être utilisé pour des modifications de champs numériques par incrément fixe. Ni *AjoutChampChoix*, ni l'éditeur à choix multiples ne connaissent ou s'intéressent au type du champ à modifier. La seule limite qu'ils imposent est que le champ ne doit pas dépasser 10 octets. Par exemple, vous pouvez incrémenter ou décrémenter un champ de type entier par multiple de 100 plutôt que par 1 (100 * *Facteur* plutôt que 1 * *Facteur*).

Le programme exemple SAISIE.PAS illustre l'utilisation des champs à choix multiples (types énumérés et types numériques).

Ecrans de saisie de données imbriqués

TPSAISIE offre deux possibilités pour traiter des écrans de saisie supérieurs à la taille de l'écran physique. La première est une interface qui vous permet de créer des écrans de saisie imbriqués. Supposons que l'enregistrement d'une grande base de données contienne plusieurs champs sur des données relatives à l'historique des emplois d'une personne. Plutôt que d'afficher tous ces champs dans l'écran principal de saisie, vous pouvez afficher un champ fictif appelé 'Historique employé'. Lorsque l'utilisateur place le curseur dans ce champ et presse (ENTER), un deuxième écran de saisie de données contenant tous les champs relatifs au sujet pourrait être affiché dans une fenêtre popup. Lorsque l'utilisateur a terminé de saisir ses données dans le deuxième écran, vous pouvez alors reprendre les modifications dans l'écran de saisie principal.

La création d'écrans de saisie de données imbriqués se fait en trois étapes. Le première est l'ajout d'un champ fictif dans l'écran principal de saisie en utilisant *AjoutChampEmboite*. La deuxième est de créer un deuxième écran de saisie et d'écrire une procédure qui gère les appels à *EcranEdit* pour elle. Et, la troisième est de modifier le code qui exécute les commandes de sortie renvoyées par *EcranEdit* lorsque les champs de l'écran de saisie principal ont été modifiés. Si la commande de sortie est *ESemboite*, vous pourrez appeler la procédure que vous avez écrite à l'étape 2. Pour un exemple d'utilisation, voir SAISIE.PAS.

L'interface fournie pour les écrans de saisie emboîtés peut être utilisée ailleurs. Par exemple, supposez qu'un champ d'un enregistrement de données est un buffer à part appelé "Notes" et que vous désirez utiliser l'éditeur binaire de Borland (inclus dans la *ToolBox Editor*) pour modifier les notes. Le meilleur moyen pour effectuer cette tâche est de créer un champ *Notes*

fictif dans l'écran de saisie en utilisant *AjoutChampEmboite* et d'appeler l'éditeur lorsque la commande *ESemboite* est renvoyée.

Les clés du succès dans ce cas sont les champs *Buffer* et *EOText* dans le bloc de contrôle de l'éditeur (EdCB). Au lieu de charger normalement le fichier dans l'éditeur, faites ce qui suit :

- Copiez le texte du champ *Notes* dans le buffer de l'éditeur
- Spécifiez la taille du texte placé dans le buffer
- Ajoutez un ^Z à la fin du texte

Le code pourrait ressembler à cela :

```
Move(MonEnreg.Notes, MonEdCB.Buffer^, MonEnreg.OctetsADeplacer);
MonEdCB.EOText := MonEnreg.OctetsADeplacer;
{Buffer pointe sur un tableau indicé à partir de 0}
MonEdCB.Buffer^[MonEnreg.OctetsADeplacer] := ^Z;
```

Lorsque vous devriez normalement sauvegarder le fichier modifié, vous renverserez le processus :

```
MonEnreg.OctetsADeplacer := MonEdCB.EOText;
Move(MonEdCB.Buffer^, MonEnreg.Notes, MonEnreg.OctetsADeplacer);
```

Déroulement d'écran de saisie de données

La deuxième façon de travailler avec des écrans de saisie supérieurs à la taille de l'écran est de spécifier une fenêtre d'entrée de données en utilisant *InitFenEnt*. Cette procédure vous permet de spécifier une partie de l'écran qui sera utilisée pour afficher l'écran de saisie de données. Supposons que la fenêtre contient 15 lignes mais que vous avez ajouté des champs avec des coordonnées de lignes telles que 50. Le programme de gestion de l'écran déroulera alors la fenêtre autant de fois que le curseur sera placé dans un champ qui n'est pas actuellement affiché dans la fenêtre.

Quelques points clés méritent d'être gardés en mémoire lors de l'utilisation de cette possibilité. Premièrement, la fenêtre d'entrée de données peut être déroulée verticalement (en haut et en bas) mais pas horizontalement (à gauche et à droite). Vous devez par conséquent vous assurer que tous les champs peuvent être affichés horizontalement. Deuxièmement, bien qu'en général le message associé à un champ modifié se trouve sur une ligne différente, ce n'est pas toujours le cas lors du déroulement d'un écran de saisie. Pour éviter une perte de temps, le programme de gestion d'écran suppose qu'il n'y a pas de message sur une ligne située avant le premier champ à modifier ni au-dessous du dernier champ. Ceci n'a pas d'importance si l'écran n'est pas déroulé entièrement.

Troisièmement, souvenez-vous que les coordonnées sont toujours traitées en mode relatif par rapport à la fenêtre d'entrée de données lors de l'ajout d'un champ dans l'écran tout entier. Lorsque *InitEnregES* est appelée, elle

initialise les coordonnées par défaut de la fenêtre à l'écran tout entier. Aussi, si *InitFenEnt* n'est jamais appelée, les coordonnées du champ seront effectivement traitées en mode absolu tant que les champs seront contenus dans l'écran. Mais si vous exécutez par exemple l'instruction :

```
InitFenEnt(MonEES, 1, 5, 80, 20, Encadre, TextAttr, TextAttr);
```

les coordonnées de tout champ ajouté à *MonEES* seront traitées relativement à la ligne 5, colonne 1 de l'écran physique.

Comme les fenêtres popup créées par TPFENETR, les fenêtres de saisie de données peuvent comporter ou non un cadre. Si la fenêtre comporte un cadre et qu'un support de souris a été activé par un appel à *ActiveSaisieSouris*, TPSAISIE utilisera le bord droit du cadre comme une barre de défilement en supposant que l'écran de saisie de données est plus large que la fenêtre. Si vous placez le pointeur de la souris dans le coin supérieur droit de la fenêtre et que vous cliquez à gauche, l'écran se déroulera d'un ligne vers le haut. Si vous cliquez le bouton gauche sur le coin inférieur droit, l'écran se déroulera d'une ligne vers le bas. Des déplacements plus rapides d'un zone à une autre peuvent être obtenus en plaçant le pointeur de la souris sur la position appropriée de l'ascenseur et en cliquant à gauche.

Affichage des messages d'aide

Comme TPSAISIE vous permet de spécifier la routine à appeler lorsque le champ a été modifié (voir les explications sur les champs calculés ci-dessus), vous pouvez spécifier une routine qui peut être appelée juste avant les modifications. Bien qu'il y ait plusieurs constructions possibles d'une telle routine, le message d'aide le plus évident à afficher est d'expliquer à l'utilisateur ce qu'il doit entrer dans ce champ. Tout ce que vous avez à faire est d'appeler *InitPtrAvantEdit* avec le nom de votre enregistrement d'écran de saisie et l'adresse de votre routine. La routine elle-même pourrait être de la forme suivante :

```
{$F+}
procedure AffichMessAide(var ESE : ESEnreg);
begin
   case ESE.IDCourant of
      {Affichage du message d'aide du champ}
   end;
end;
{$F-}
```

Voir SAISIE.PAS comme exemple de routine affichant les messages d'aide de cette façon.

Gestion des erreurs

Les messages d'erreurs fournis par TPSAISIE peuvent être traités de la même façon. Pour mettre en oeuvre un gestionnaire d'erreurs, appelez *InitPtrErreur* avec le nom de votre enregistrement d'écran de saisie et l'adresse de la routine qui affichera les messages d'erreurs. A chaque fois qu'une erreur se produira, TPSAISIE passera un code erreur et un message à la routine comme 'Nombre hors limite' ou 'Format non valide'. Le code erreur indique le type de l'erreur (voir la liste des constantes dans TPSAISIE.PAS appelée "Codes des messages d'erreurs"), vous permettant de substituer vos propres messages d'erreurs à la place des messages par défaut.

Votre gestionnaire d'erreurs pourrait avoir la forme suivante :

```
{$F+}
procedure GestErreurs(var ESE : ESEnreg; Code : Byte; Msg : String);
begin
   {Affichage du message d'erreur et attente
    de la frappe d'une touche}
   {Efface la partie de l'écran recouverte par le message}
end;
{$F-}
...
InitPtrErreur(ESE, @GestErreurs);
```

Si le gestionnaire d'erreurs n'est pas installé et qu'une erreur se produit, TPSAISIE activera la cloche et reprendra ensuite l'exécution si l'erreur était due à une donnée non valide ou s'arrêtera s'il s'agissait d'une erreur fatale (violation mémoire).

Voir SAISIE.PAS comme exemple de routine affichant des messages d'erreur.

Tâches d'arrière-plan

Comme de nombreuses autres unités de *Turbo Professionnel*, TPSAISIE facilite l'exécution de tâches d'arrière-plan pendant l'attente de la frappe d'une touche du clavier. Voir l'appendice C pour plus de détails et SAISIE.PAS en exemple.

Aide contextuelle

Bien que l'aide contextuelle ne soit pas une caractéristique inhérente à TPSAISIE, elle fournit des interfaces vous permettant d'implémenter facilement des systèmes d'aide sophistiqués. La première étape est d'assigner chaque champ d'un écran de saisie de données avec son propre numéro (1-10000), de passer ce nombre comme paramètre d'index d'aide à la routine qui ajoute ce champ à l'écran de saisie. Les autres étapes sont

expliquées en annexe C, SAISIE.PAS montre comment utiliser harmonieusement TPSAISIE et TPAIDE.

Support de la souris

Comme les unités orientées écran de *Turbo Professionnel,* TPSAISIE fournit un support interne de la souris. Pendant la saisie de données, les fonctions suivantes sont traitées par la souris :

ESsouris <Bouton gauche>
Sélectionne un nouveau champ utilisant la souris. Cette commande vous permet de vous déplacer rapidement d'un champ à un autre sur l'écran de saisie de données. Placez simplement le pointeur de la souris sur le champ désiré puis cliquez le bouton gauche de la souris. Si le champ de destination comporte un écran de saisie de données imbriqué, *EcranEdit* sortira avec le code de retour de *ESemboite.* Si le champ courant est un champ à choix multiples, la valeur sera incrémentée si vous déplacez le pointeur de la souris sur celui-ci et que vous cliquez le bouton gauche. Si le champ courant est un booléen ou un champ Oui/Non, la valeur sera basculée si vous y placez le pointeur de la souris et que vous cliquez le bouton gauche. Finalement si le champ courant a été spécialement modifié (en appelant *InitSortieSurSecondClic*), *EcranEdit* sortira avec le code de retour de *ESsortieParClic* si vous placez le pointeur de la souris sur ce champ et que vous cliquez le bouton gauche (le programme exemple SAISIE.PAS illustre ces caractéristiques. Voir la documentation du programme au chapitre 13).

ESquitte <Bouton droit>
Arrêt. Identique à la frappe de (ESC).

ESaide <Les deux boutons>
Aide. Identique à la frappe de (F1).

ESchampSuiv <Bouton central>
Identique à la frappe de (ENTER) (pour les souris à trois boutons seulement).

Remarquez que les fonctions de la souris peuvent être traitées comme des affectations de touches. Voir appendice C pour plus de détails.

Maintenant, comment implémenter la souris dans un programme qui utilise TPSAISIE ? Nous supposons que vous avez déjà lu la documentation de l'unité TPSOURIS (sinon faites-le s'il-vous-plaît) qui explique les principes généraux relatifs à l'utilisation de la souris.

Tout d'abord, vous devrez sélectionner un pointeur de souris software au début de votre programme car TPSAISIE utilise le pointeur hardware pour ses éditeurs de champs. Deuxièmement, vous devrez appeler *ActiveSaisieSouris*. Enfin, si vous écrivez votre propre routine de saisie clavier afin d'implémenter une tâche d'arrière-plan, vérifiez que vous avez suivi les conseils de l'appendice C.

Le meilleur moyen d'apprendre à utiliser TPSOURIS et TPSAISIE conjointement est d'étudier le code source du programme de démonstration SAISIE.PAS. Comme TPSAISIE elle-même, SAISIE utilise une directive de compilation conditionnelle qui se trouve dans TPDEFINE.INC, 'UseMouse' qui détermine si le support de la souris est implémenté ou non. En utilisant un éditeur ou un programme qui permet de "browser" un fichier, vous pouvez retrouver toutes les occurrences de cette directive afin de voir précisément ce qu'il y a à faire pour ajouter le support de la souris au programme.

Types définis par l'utilisateur

Bien que TPSAISIE offre un support interne exceptionnel pour de nombreux types de données, vous aurez parfois à utiliser des types de données qu'elle ne supporte pas directement. Heureusement, TPSAISIE vous permet de définir des types utilisateurs. La mise en oeuvre nécessite trois étapes.

Tout d'abord, vous devrez écrire une routine pour convertir votre type de données en une chaîne qui peut être affichée et modifiée. Deuxièmement, vous devrez écrire une routine de contrôle de saisie (ces deux aspects sont expliqués ci-dessous). Et finalement, vous devrez appeler *AjouteChampUtl* pour ajouter un champ de ce nouveau type à votre écran de saisie.

Routines de conversion

Une routine de conversion est une procédure qui convertit une variable d'un type particulier en une chaîne et réciproquement. TPSAISIE comporte des routines de conversion pour tous les types de données qu'elle supporte. Mais si vous devez utiliser TPSAISIE avec des types définis par l'utilisateur vous devrez écrire vos propres routines de conversion. Nous n'expliquerons pas cela complètement mais vous pouvez étudier le code source du programme TPSAISIE afin d'apprendre à écrire ces routines. Une des routines au moins devrait vous servir de modèle. Néanmoins nous vous conseillons d'adopter la structure générale suivante pour écrire vos propres routines de conversion :

```
{$F+}
procedure RoutineConversion(var EC : EnrChamp; PostEdit : Boolean);
begin
   if PostEdit then
      {Convertit EC.EditeChn^ en EC.PtrVar^}
   else
      {Convertit EC.PtrVar^ en EC.EditeChn^} ;
end;
{$F-}
```

Le paramètre *EC* est un enregistrement contenant toutes les données sur le champ. Le paramètre *PostEdit* vous indique si vous devez convertir la variable en une chaîne ou réciproquement.

Routines de contrôle

Une routine de contrôle est assez proche d'une routine de conversion mais son objet est de contrôler les données saisies. Une telle routine pourrait déterminer si un nombre appartient bien à un intervalle donné, si un champ est partiellement vide alors qu'il doit être rempli ou si une chaîne appartient à une liste de réponses valides.

Là encore, nous ne pouvons expliquer complètement comment écrire des routines de contrôle, aussi nous vous recommandons d'étudier les exemples du code source. Deux routines de contrôle spéciales, *VideOuRempli* et *BlancsInterdits* sont d'un intérêt particulier. Dans tous les cas, une routine de contrôle devra avoir la forme suivante :

```
{$F+}
procedure ControleValidite(var EC : EnrChamp;
                   var ErrCode : Byte;
                   var ChnErreur : PtrChaine) : Boolean;
begin
   {Supprime les caractères de format étrangers de FR.EditeChn^}
   {Convertit la chaine en son type approprié}
   {Vérifie la validité du résultat}
end;
{$F-}
```

La structure réelle, quoi qu'il en soit, dépendra du type de donnée.

Modification du rôle des touches

Il y a plusieurs façons de modifier l'affectation des touches aux commandes de l'éditeur. Voir l'appendice C pour plus de détails.

Il est également possible de désactiver des commandes données sans modifier l'affectation des touches. Les constantes typées globales, *CommandesChaine*, *Commandes* et *ChoixCommandes* contiennent l'ensemble des commandes qui sont valides pour chacun des trois éditeurs, vous pouvez désactiver une commande simplement en modifiant l'ensemble approprié.

Compilation de TPSAISIE

Avant de compiler TPSAISIE, n'oubliez pas ce qui suit. Tout d'abord, deux catégories de routines sont normalement désactivées via les directives de compilation conditionnelle : celles qui travaillent avec les réels 8087 (single, double, extended, comp) et celles qui travaillent avec des réels BCD (implémentées dans TPBCD). Il y a aussi deux catégories de routines qui peuvent facilement être désactivées si vous voulez réduire la taille d'une unité. Celles travaillant avec les champs à choix multiples et celles relatives à l'éditeur numérique (de droite à gauche). Finalement, il existe une directive, définie par défaut qui active ou désactive le support de la souris.

Pour activer ou désactiver l'une de ces caractéristiques, vous devrez modifier l'un des deux fichiers : TPSAISIE.PAS (pour ajouter le support des réels 8087) ou TPDEFINE.INC (pour modifier n'importe quelle autre directive). Vous pouvez désactiver une caractéristique particulière en insérant un point entre l'accolade gauche et le signe $, par exemple :

```
{.$Define IncludeNumeric}
```

ou activer cette caractéristique en enlevant le point. Par défaut, l'éditeur numérique et l'éditeur à choix multiples sont activés alors que le support des réels BCD 8087 est désactivé.

Routines non documentées

Les routines suivantes de bas niveau sont interfacées avec TPSAISIE mais ne sont pas documentées. Elles facilitent le travail des programmeurs avertis qui ont besoin de routines spéciales de contrôle et de conversion pour gérer des types de données non prévus. Vous pourrez vous inspirer du code source de l'unité TPSAISIE pour développer les routines dont vous aurez besoin.

```
procedure AjoutChampPrim(var ESE : ESEnreg; var EC : EnrChamp;
                         var ChnFormat : String; var ChnMess : string);
   {-Routine de bas niveau pour allouer et finir d'initialiser
    un EnrChamp}

function ChercheIDChamp(var ESE : ESEnreg; IDChamp : Byte) : PtrEnrChamp;
   {-Retourne un pointeur vers EnrChamp avec l'ID spécifiée, ou Nil si
    introuvable}
```

```
procedure InitIndicsFormat(var Format :String; var IndicF :IndicsFormat);
   {-Initialise une table d'indicateurs de format}

procedure FusionFormat(Format, Chn : String; var S : String;
                       IndicEdit : Word);
   {-Fusionne Format et Chn, puis retourne le résultat dans S}

procedure ReduitFormat(var Format, EditeChn, S : String);
   {-Supprime les caractères de format de EditeChn
    et retourne le résultat dans S}
```

TPSAISIE : Déclarations

Types

```
CarEnMajOuMinusc = (SansChangement, Majusc, Minusc);
```

Ces variables définissent la façon dont sont traités les caractères des masques de saisie définis par l'utilisateur. Voir la partie relative aux "Masques de saisie" au début de ce chapitre pour plus de détails.

```
EnsembleCarac = Set of Char;
```

Les variables de ce type contiennent typiquement un ensemble de caractères qui correspondent à un caractère particulier d'un masque de saisie et sont utilisées pour valider les données saisies au clavier.

```
EnsembleCommandes = set of TypeEs;
```

Une variable de ce type contient l'ensemble des commandes qui sont implémentées par l'un des trois éditeurs de champs.

```
EsEnreg =
   record
   ...
      IdCourant : Word; {ID du champ courant}
   end;
```

Les variables de ce type sont utilisées pour enregistrer des données sur un écran de saisie particulier. Les emplacements mémoire des différents champs, le nombre de champs et ainsi de suite. Le seul champ de la structure de données que vous utiliserez probablement est *IDCourant*. Il contient l'identificateur du champ traité actuellement par le programme de gestion de la saisie. Les différentes utilisations de cette donnée sont expliquées ailleurs dans ce chapitre.

```
TypeEs = EsRien..EsSouris;
```

Une variable de ce type est généralement utilisée pour enregistrer le résultat d'un appel à *EcranEdit* qui renvoie toujours une valeur de type byte dans l'intervalle *EsRien..EsQuitte* (Voir la partie des constantes ci-dessous).

```
EnrChamp =
   Record
      IDChamp : Word;
      ...
      Format   : PtrChaine;
      EditeChn : PtrChaine;
      ....
      LimiteInf : TypeIntervalle;
      LimiteSup : TypeIntervalle;
      ....
      PtrVar : Pointer;
      ...
   end
PtrEnrChamp = ^EnrChamp;
```

Les variables de type *EnrChamp* contiennent toutes les données sur un champ particulier. La plupart des programmeurs n'auront jamais besoin de connaître ces structures de données. L'exception la plus probable concerne les programmeurs avertis qui ont besoin d'écrire des routines spéciales de conversion et de contrôle pour traiter les types de données non prévus. La variable *IdChamp* est le numéro unique affecté au champ lorsqu'il est ajouté à un écran de saisie. *Format* pointe sur le masque de saisie du champ. *EditeChn* pointe sur la chaîne qui est utilisée lors de la modification du champ. *LimiteInf* et *LimiteSup* contiennent éventuellement les limites inférieure et supérieure du champ et *PtrVar* pointe sur la variable en cours de modification.

```
TypeModeEnroule = (ArretAuxBords, EnrouleAuxBords, SortieAuxBords);
```

Voir la procédure *InitModeEnroul* plus loin dans ce chapitre pour l'explication des valeurs qui composent ce type énuméré.

Constantes

```
ChoixCommandes : EnsembleCommandes = [EsCarac, EsRestaure,
EsGauche..EsMotDroite, EsAide..EsSouris];
```

Ces variables contiennent l'ensemble des commandes d'édition qui sont supportées par l'éditeur à choix multiples. Elles servent seulement à désactiver une commande particulière en l'enlevant de l'ensemble. Vous ne devrez jamais ajouter d'élément à cet ensemble à moins que vous ne vouliez y replacer une commande désactivée précédemment.

```
CaracVirgule : Char = ',';
```

Cette variable contient le caractère qui est affiché lorsqu'un séparateur (',') est nécessaire dans un champ numérique. Voir la partie relative aux "Masques de Saisie" au début de ce chapitre pour plus de détails.

```
ChnGaucheMonnaie : string[5] = '$';
ChnDroitMonnaie : string[5] = '';
```

Ces chaînes contiennent des symboles monétaires pour un pays donné. Voir la partie sur "Conventions et Monnaies Internationales" pour plus de détails.

```
CaracDecimal : Char = '.';
```

Cette variable contient le caractère qui est affiché lorsqu'un point décimal est nécessaire dans un champ numérique. Voir la partie sur "Masques de Saisie" au début de ce chapitre pour plus de détails.

```
PtrAideSaisie : Pointer = nil;
```

Si cette variable ne contient pas nil, elle doit pointer sur une routine utilisateur qui gère les demandes d'aide. Voir la partie sur "Aide Contextuelle" au début de ce chapitre pour plus de détails.

```
SaisieEgaleCtrl : Boolean = True;
```

Si cette variable vaut "True", tous les caractères de contrôle d'un champ sont affichés en caractères alphabétiques (^G comme 'G' par exemple). Si elle vaut "False", les caractères de contrôle sont affichés normalement.

```
EsRien          = 00;     {n'est pas une commande}
EsEnrSuiv       = 26;     {enregistrement suivant}
EsEnrPreced     = 27;     {enregistrement précédent}
EsEmboite       = 30;     {sortie de structure imbriquée}
EsUtl0          = 31;     {touche de sortie définie par l'utilisateur}
EsUtl9          = 40;
EsFinSaisie     = 41;     {édition complètement terminée}
EsQuitte        = 42;     {restaure la chaîne par défaut et sortie}
EsSortieParClic = 43;     {sortie sur un double clic souris}
EsSouris        = 44;     {sélectionne le champ pointé par la souris}
```

Chacune de ces constantes correspond à une commande de modification de déplacement du curseur ou de sortie. Parmi ces commandes, toutes sauf *EsRien* et *EsSouris* appartiennent à la catégorie des commandes de sortie, et ce sont probablement les seules dont vous pouvez avoir besoin. *EsUtl0..EsUtl9* sont de simples interfaces qui permettent au programmeur de définir de nouvelles commandes de sortie. La commande *EsEmboite* est retournée lorsque le champ courant est désigné comme devant être lié à un écran de saisie de données imbriqué et que l'utilisateur a exécuté la commande *EsChampSuiv* (normalement affectée à (ENTER)). La commande *EsSortieParClic* est expliquée dans la procédure *InitSortieSurSecondClic*. Le programme exemple SAISIE.PAS illustre leur utilisation. Voir la partie sur "Commandes d'Edition" au début de ce chapitre pour la description de toutes les autres commandes disponibles.

```
MetEnMajuscule : array[Utl1..Utl8] of CarEnMajOuMinusc = (
   SansChangement, SansChangement, SansChangement, SansChangement,
   SansChangement, SansChangement, SansChangement, SansChangement);
```

Les valeurs de ce tableau correspondent aux caractères des masques de saisie spéciaux *Utl1..Utl8*. Voir la partie sur "Masques de Saisie" au début de ce chapitre pour plus de détails sur l'utilisation de ce tableau.

```
ErreurInit              = 0;    {enregistrement Es non initialisé}
ErreurDepassement       = 1;    {trop de champs}
ErreurParam             = 2;    {paramètre incorrect dans une procédure}
ErreurMemoire           = 3;    {mémoire insuffisante}
ErreurFormat            = 4;    {format non valide}
ErreurIntervalle        = 5;    {donnée saisie hors des bornes de
                                 l'intervalle}
ErreurChpObligatoire    = 6;    {champ obligatoire}
ErreurBlancs            = 7;    {le champ contient des blancs}
ErreurPartiel           = 8;    {le champ est partiellement rempli}
ErreurCaracNonValide    = 9;    {le champ contient des caractères non
                                 valides}
```

Ce sont les codes erreur qui pourraient être passés à un gestionnaire d'erreur défini par l'utilisateur et activé par *InitPtrErreur*. Remarquez que les erreurs 0..3 sont considérées comme des erreurs fatales et entraînent l'arrêt du programme dès que le gestionnaire d'erreur (éventuellement) est appelé.

```
ModeInsert : Boolean = True;
```

Cette variable indique si les éditeurs de champs sont en mode insertion ou recouvrement, elle permet à l'utilisateur de définir son propre gestionnaire clavier (pointé par *PtrToucheSaisie*) pour afficher le mode courant.

```
Commandes : EnsembleCommandes =
   [EsCarac, EsRestaure..EsDelLigne, EsIns..EsSouris];
```

Cette variable contient l'ensemble des commandes d'édition qui sont supportées par l'éditeur numérique. Elle permet de désactiver une commande donnée en l'enlevant de cet ensemble. Vous ne devrez jamais ajouter de commande à cet ensemble sauf si c'est une commande que vous avez précédemment désactivée.

```
Off = False;
On = True;
```

Ces deux constantes vous aideront à écrire un code auto-documenté avec TPSAISIE. Elles sont généralement utilisées comme paramètres des procédures qui changent certaines options d'édition en On ou Off.

```
CommandesLectSeulement : EnsembleCommandes =
   [EsHome..EsMotDroite, EsIns..EsBackTab, EsChampSuiv..EsSouris];
```

Cette variable contient l'ensemble des commandes d'édition qui sont disponibles en mode lecture seulement (voir la procédure *EcranEdit*), cet

ensemble de commandes est également utilisé lorsque le champ courant sert de champ de liaison avec un écran de saisie de données imbriqué (voir la procédure *AjoutChampEmboite* pour des explications). Elle contient toutes les commandes non destructives (celles qui ne modifient pas le contenu d'un champ).

```
CommandesChaine : EnsembleCommandes = [EsCarac..EsSouris];
```

Cette variable contient l'ensemble des commandes d'édition qui sont supportées par l'éditeur chaîne. Elle permet de désactiver une commande particulière en l'enlevant de cet ensemble. Vous ne devrez jamais ajouter de commandes à cet ensemble sauf si c'est une commande que vous avez précédemment désactivée.

```
Utl1 = '1';
Utl2 = '2';
Utl3 = '3';
Utl4 = '4';
Utl5 = '5';
Utl6 = '6';
Utl7 = '7';
Utl8 = '8';
```

Ces caractères spéciaux peuvent être utilisés dans des masques de saisie pour contrôler la saisie utilisateur par rapport à des ensembles de caractères (*Ens1Utl..Ens8Utl*). Voir la partie sur "Masques de Saisie" au début de ce chapitre pour plus de détails sur leur utilisation.

Ces ensembles de caractères correspondent aux masques de saisie spéciaux, *Utl1..Utl8*. Voir la partie sur "Masques de Saisie" au début de ce chapitre pour plus de détails sur leur utilisation.

Variables

```
PtrToucheSaisie : Pointer;
```

Cette variable pointe sur une routine utilisateur qui gère les entrées clavier. Voir la partie sur "Tâches d'arrière-plan" au début de ce chapitre pour plus de détails sur leur utilisation.

ActiveSaisieSouris

Déclaration

```
procedure ActiveSaisieSouris;
```

Objet

Active la souris dans TPSAISIE.

Commentaires

Cette routine active le support de la souris pour TPSAISIE. En plus de son utilisation par les autres routines, elle active le gestionnaire d'événements de la souris de TPSOURIS et initialise *PtrToucheSaisie* pour pointer sur la routine *LitToucheOuBouton* de TPSOURIS. Voir l'introduction de l'unité TPSOURIS pour plus de détails.

Voir aussi

DesactiveSaisieSouris (TPSAISIE)

AjoutChampArray

Déclaration

```
procedure AjoutChampArray(var ESE : EsEnreg;
                          Mess : string;
                          PLigne; PCol : byte;
                          Format : string;
                          eLigne, eCol, LongEdit : byte;
                          IndexAide : Word;
                          Validation : Pointer;
                          var EditVar);
```

Objet

Ajoute un champ de type array (2 caractères) à l'écran de saisie.

Commentaires

Cette procédure est destinée principalement aux programmeurs de base de données qui utilisent fréquemment un *array[1..n] of Char*, alors que les programmeurs Pascal utiliseraient une string[n].

Les paramètres de *AjoutChampArray* sont les mêmes que ceux de *AjoutChampString* sauf le dernier. *EditVar* doit être un *array[1..n] of Char*, où *n* est la plus grande des deux valeurs *LongEdit* ou *Length(Format)*. Voir la partie sur *AjoutChampString* pour une explication des autres paramètres.

Notez que l'option de suppression de blancs est automatiquement désactivée pour les champs de ce type afin de garantir que les tableaux soient complétés sur toute leur longueur avec des blancs.

Exemple

```
var
   Tableau : array[1..8] of char;
...
AjoutChampArray(ESE, 'Tableau : ', 1, 1, '', 10, 8, 0, Nil, Tableau);
```

Ajoute un champ de tableau de caractères qui sera affiché dans le coin supérieur gauche de l'écran.

Voir aussi

AjoutChampString (TPSAISIE)

AjoutChampBcd

Déclaration

```
procedure AjoutChampBcd(var ESE : EsEnreg;
                        Mess : string;
                        PLigne; PCol : byte;
                        Format : string;
                        eLigne, eCol : byte;
                        IndexAide : Word;
                        BcdInf, BcdSup : BCD;
                        Position : byte;
                        var EditBcd :BCD);
```

Objet

Ajoute un champ de type BCD à un écran de saisie.

Commentaires

Cette routine ajoute un champ BCD à l'écran de saisie spécifié. Ces paramètres sont identiques à ceux des autres champs ou des autres types, flottants.

ESE est l'écran de saisie auquel vous ajoutez le champ BCD. *Mess* est le message à afficher à gauche (habituellement) du champ d'édition. *PLigne, PCol* sont les coordonnées écran du message, *Format* est le masque de saisie du champ. Remarquez que ce paramètre est obligatoire pour les champs BCD. *eLigne* et *eCol* sont les coordonnées écran pour le champ à éditer. *IndexAide* est l'index à passer à la routine d'aide définie par l'utilisateur, si elle existe, lorsque (F1) est pressée. *BcdInf* et *BcdSup* représentent les valeurs minimum et maximum pour le champ BCD à modifier. S'ils sont égaux, aucun contrôle d'intervalle ne sera exécuté. *Position* est le nombre maximum de positions décimales à afficher. Cette valeur est ignorée si le masque de saisie contient un point décimal fixe ('.'). *EditBcd* est la variable BCD à modifier.

Remarques : Afin d'utiliser *AjoutChampBcd,* vous devez activer la directive de compilation conditionnelle *UseBcd* dans TPDEFINE.INC et recompiler TPSAISIE.

Exemple

```
var Salaire : BCD;
...
AjoutChampBCD(ESE, 'Salaire', 1, 1, '999 999,99 F', 1, 8, 1, ZeroBCD,
              ZeroBCD, 0, Salaire);
```

Salaire est un champ de type BCD à afficher dans le coin supérieur gauche de l'écran. Aucun contrôle d'intervalle ne sera exécuté bien qu'il y ait une valeur maximum de 999 999,99 et une valeur minimum de 0 induite par le

contenu du masque de saisie. Le paramètre *Position* est inopérant puisque les positions décimales sont fixes dans le masque de saisie.

```
var Gains : BCD;
...
AjoutChampBCD(ESE, 'Gains', 1, 1, '###########', 1, 10, 1, ZeroBCD,
              ZeroBCD, 2, Gains);
```

Aucun contrôle spécial d'intervalle ne sera exécuté sur ce champ bien qu'il y ait une valeur maximum de 99 999 999 999 et une valeur minimum de -9 999 999 999 induite par la largeur et le contenu du masque de saisie. Le paramètre position est important car il n'y a pas de position décimale fixe dans le masque de saisie.

AjoutChampBoolean

Déclaration

```
procedure AjoutChampBoolean(var ESE : EsEnreg;
                            Mess : string;
                            PLigne; PCol : byte;
                            Format : string;
                            eLigne, eCol : byte;
                            IndexAide : Word;
                            var EditBool: Boolean);
```

Objet

Ajoute un champ de type booléen à l'écran de saisie.

Commentaires

Cette routine ajoute un champ booléen à l'écran de saisie spécifié. Ces paramètres sont identiques à ceux des routines qui ajoutent des champs numériques mais cette routine ne comporte aucun paramètre pour préciser les valeurs minimum et maximum et le paramètre de masque de saisie est facultatif (par défaut, un masque de saisie de 'B' est pris en compte).

ESE est l'écran de saisie auquel vous ajoutez le champ booléen. *Mess* est le message à afficher à gauche (habituellement) du champ à modifier. *PLigne* et *PCol* sont les coordonnées écran du message. *Format* est le masque de saisie pour le champ. Remarquez que ce paramètre est facultatif pour les champs booléens, s'il est spécifié, il devra contenir seulement des caractères littéraux (facultatifs) et la lettre 'B'. *eLigne* et *eCol* sont les coordonnées écran du champ à modifier. *IndexAide* est l'index à passer à la routine d'aide utilisateur si elle existe lorsque (F1) est pressée. *EditBool* est la variable booléenne à modifier.

Exemple

```
var Marie : Boolean;
...
AjoutChampBoolean(ESE, 'Marié(e)', 1, 1, '', 1, 9, 1, Marie);
```

Marie est un champ de type booléen à afficher dans le coin supérieur gauche de l'écran. Remarquez que le format est vide, ainsi le masque de 'B' sera pris en compte.

Voir aussi

AjoutChampOuiNon (TPSAISIE)

AjoutChampByte

Déclaration

```
procedure AjoutChampByte(var ESE : EsEnreg; Mess : string;
                         pLigne, pCol : byte ; format : string;
                         eLigne, eCol : byte; IndexAide : Word;
                         byteInf, byteSup : byte;
                         var Editbyte : byte);
```

Objet

Ajoute un champ de type byte à un écran de saisie.

Commentaires

Cette routine ajoute un champ byte à l'écran spécifié. Ces paramètres sont identiques à ceux des routines qui ajoutent des champs des autres types entiers (ShortInt, Integer, Word, LongInt) et pour les champs de type Char.

ESE est l'écran de saisie auquel vous ajoutez le champ byte. *Mess* est le message à afficher à gauche (habituellement) du champ à modifier. *PLigne* et *PCol* sont les coordonnées écran du message. *Format* est le masque de saisie de ce champ. Remarquez que ce paramètre est obligatoire pour les champs byte. *eLigne* et *eCol* sont les coordonnées écran du champ à modifier. *IndexAide* est l'index à passer à la routine d'aide utilisateur si elle existe lorsque (F1) est pressée. *ByteInf* et *ByteSup* représentent les valeurs minimum et maximum pour le champ byte à modifier. S'ils sont égaux aucun contrôle d'intervalle n'est exécuté. *Editbyte* est la variable byte à modifier.

Exemple

```
var Enfants : Byte;
...
AjoutChampByte(ESE, 'Enfant(s)', 1, 1, '99', 1, 10, 1, 0, 20,
               Enfants);
```

Enfants est un champ de type Byte à afficher dans le coin supérieur gauche de l'écran. Remarquez que les paramètres *ByteInf* et *ByteSup* exigent que la valeur appartienne à l'intervalle 0..20. Si ces deux paramètres sont égaux la valeur maximum sera 99 comme indiquée par le masque de saisie.

AjoutChampChar

Déclaration

```
procedure AjoutChampChar(var ESE : EsEnreg;
                         Mess : string;
                         pLigne, pCol : byte ;
                         format : string;
                         eLigne, eCol : byte;
                         IndexAide : Word;
                         CaracInf, CaracSup : Char;
                         var EditChar : Char);
```

Objet

Ajoute un champ de type Char à un écran de saisie de données.

Commentaires

Cette routine ajoute un champ Char à l'écran de données spécifié. Ces paramètres sont identiques à ceux de *AjoutChampByte* sauf que les trois derniers paramètres sont de type Char plutôt que de type byte. Pour les explications des autres paramètres voir la procédure *AjoutChampByte*.

Voir aussi

AjoutChampByte (TPSAISIE)

AjoutChampChnDate

Déclaration

```
procedure AjoutChampChnDate(var ESE : EsEnreg;
                            Mess : string;
                            PLigne; PCol : byte;
                            Format : ChaineDate;
                            eLigne, eCol : byte;
                            IndexAide : Word;
                            var EditDate : ChaineDate);
```

Objet

Ajoute un champ de type *ChaineDate* à l'écran de saisie.

Commentaires

Cette routine ajoute un champ de type *ChaineDate* à l'écran spécifié. Elle permet d'enregistrer des dates sous forme de chaînes de caractères plutôt que sous forme de variables date.

Les paramètres passés à *AjoutChampChnDate* sont similaires à ceux de *AjoutChampChaine* mais le pointeur facultatif sur la routine de contrôle utilisateur n'est pas nécessaire. Le contrôle est toujours géré par la routine interne via la fonction *ChaineDateEnJMA* de l'unité TPDATE. Voir *AjoutChampChaine* pour des explications sur les autres paramètres.

Notez que les chaînes date doivent appartenir au même intervalle que les variables Date : 1/1/1600 à 12/31/3999.

Exemple

```
MaChaineDate : string[10];
...
AjoutChampChnDate(ESE, 'Date', 1, 1, 'jj/mm/aaaa', 1, 6, 1,
                  MaChaineDate);
```

Ajoute un champ *ChaineDate* à afficher dans le coin supérieur gauche de l'écran.

Voir aussi

AjoutChampDate (TPSAISIE)
AjoutChampChaine (TPSAISIE)
ChaineDateEnJMA (TPDATE)

AjoutChampChoix

Déclaration

```
procedure AjoutChampChoix(var ESE : EsEnreg;
                          Mess : string;
                          PLigne; PCol : byte;
                          Format : string;
                          eLigne, eCol : byte;
                          IndexAide : Word;
                          TailleDonnee : byte;
                          Increment : Pointer;
                          var EditVar);
```

Objet

Ajoute un champ à choix multiples dans un écran de saisie.

Commentaires

Cette routine ajoute un champ à choix multiples à l'écran spécifié. Ces paramètres sont similaires à ceux des routines qui ajoutent des champs des autres types sauf pour les paramètres *TailleDonnee* et *Increment* décrits ci-dessous.

ESE est l'écran de saisie auquel vous ajoutez le champ byte.

Mess est le message à afficher à gauche (habituellement) du champ à modifier.

PLigne et *PCol* sont les coordonnées écran du message.

Format est le masque de saisie de ce champ. Remarquez que ce paramètre est obligatoire pour les champs à choix multiples.

eLigne et *eCol* sont les coordonnées écran du champ à modifier.

IndexAide est l'index à passer à la routine d'aide utilisateur si elle existe lorsque (F1) est pressée.

TailleDonnee est la taille de la variable à modifier (*EditVar*). Elle doit être dans l'intervalle 1-10.

Increment est un pointeur sur une routine qui peut à la fois convertir une variable du même type que *EditVar* en chaîne et incrémenter ou décrémenter sa valeur.

EditVar est la variable à modifier.

Voir "Champs à Choix Multiples" au début de ce chapitre pour plus de détails et le programme SAISIE.PAS qui contient une routine d'incrémentation et de décrémentation.

Exemple

```
type
   TypeCouleur = (Rouge, Bleu, Jaune);
var
   Couleur : TypeCouleur;
...
{$F+}
procedure IncValeur(var Valeur; ID : Byte; Facteur : Integer;
                    S : String);
begin
   ...
end;
{$F-}
...
AjoutChampChoix(ESE, 'Couleur', 1, 1, 'XXXXXX', 1, 7, 1, 1,
                @IncValeur, Couleur);
```

Cet exemple ajoute un champ à choix multiples qui sera affiché dans le coin supérieur gauche de l'écran. Le rôle de la procédure *IncValeur* est de recalculer la variable valeur à partir de *Facteur* et de renvoyer une chaîne représentant le résultat dans *S*.

AjoutChampComp

Déclaration

```
procedure AjoutChampComp(var ESE : EsEnreg;
                         Mess : string;
                         PLigne; PCol : byte;
                         Format : string;
                         eLigne, eCol : byte;
                         IndexAide : Word;
                         CompInf, CompSup : Comp;
                         var EditComp : Comp);
```

Objet

Ajoute un champ de type Comp à un écran de saisie.

Commentaires

Cette routine ajoute un champ *Comp* à l'écran spécifié. Ses paramètres sont identiques à ceux de *AjoutChampBcd* mais les trois derniers paramètres sont de type Comp plutôt que de type BCD. Elle ne comporte pas de paramètre *Position* puisque le type *Comp* ne peut pas contenir de valeur fractionnaire. Pour une explication des autres paramètres et des exemples appropriés, voir *AjoutChampBcd*.

Remarque : Afin d'utiliser *AjoutChampComp*, vous devrez compiler TPSAISIE.PAS avec la directive de compilation {$N+}.

Voir aussi

AjoutChampBcd (TPSAISIE)

AjoutChampDate

Déclaration

```
procedure AjoutChampDate(var ESE : EsEnreg;
                         Mess : string;
                         PLigne; PCol : byte;
                         Format : ChaineDate;
                         eLigne, eCol : byte;
                         IndexAide : Word;
                         DateInf, DateSup : Date;
                         var EditDate : Date);
```

Objet

Ajoute un champ de type date à l'écran de saisie.

Commentaires

Cette routine ajoute un champ date à l'écran spécifié. Ses paramètres sont identiques à ceux des routines qui ajoutent des champs de type entier.

ESE est l'écran de saisie auquel vous ajoutez le champ date. *Mess* est le message à afficher à gauche (habituellement) du champ à modifier. *PLigne* et *PCol* sont les coordonnées écran du message. *Format* est le masque de saisie de ce champ. Remarquez que ce paramètre est obligatoire pour les champs date. *eLigne* et *eCol* sont les coordonnées écran du champ à modifier. *IndexAide* est l'index à passer à la routine d'aide utilisateur si elle existe lorsque (F1) est pressée. *DateInf* et *DateSup* représente les valeurs minimum et maximum de la date à modifier. S'ils sont égaux, aucun contrôle d'intervalle n'est exécuté aussi, la date entrée doit :

- être une date valide entre 1/1/1600 et 12/31/3999 ou
- être composée entièrement de zéros ('00/00/0000') ou d'espaces (' / / ').

EditDate est la variable Date à modifier.

Voir le chapitre 3, section C pour plus de détails sur les variables Date.

Exemple

```
var MaDate : Date;
...
AjoutChampDate(ESE, 'Date', 1, 1, 'jj/mm/aaaa', 1, 6, 1, DateMin,
               AujourdHui, MaDate);
```

Ajoute un champ *Date* affiché dans le coin supérieur gauche de l'écran. *MaDate* devra être comprise entre 1/1/1600 et la date du jour (la date du jour est donnée par la fonction *Aujourdhui* de l'unité TPDATE).

Voir aussi

AjoutChampChnDate (TPSAISIE) Aujourdhui (TPDATE)

AjoutChampDbl

Déclaration

```
procedure AjoutChampDbl (var ESE : EsEnreg; Mess : string;
                         pLigne, pCol : byte;
                         Format : string; eLigne, eCol : byte;
                         IndexAide : Word;
                         DblInf, DblSup : Double; Positions : byte;
                         var EditDbl : Double);
```

Objet

Ajoute un champ de type Double à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type Double à l'écran spécifié. Ces paramètres sont identiques à ceux de *AjoutChampBcd* mais les quatre derniers paramètres sont de type Double et non de type BCD. Pour des explications sur les autres paramètres et des exemples appropriés, voir *AjoutChampBcd*.

Remarque : Afin d'utiliser *AjoutChampDbl*, vous devrez compiler TPSAISIE.PAS avec la directive de compilation {$N+}.

Voir aussi

AjoutChampBcd (TPSAISIE)

AjoutChampEmboite

Déclaration

```
procedure AjoutChampEmboite(var ESE : EsEnreg;
                            Mess : string;
                            PLigne; PCol : byte;
                            Format : string;
                            eLigne, eCol, LongEdit : byte;
                            IndexAide : Word);
```

Objet

Ajoute un champ relié à un écran de saisie imbriqué.

Commentaires

Cette routine vous permet d'ajouter un champ relié à un deuxième écran de saisie imbriqué. Un champ de ce type n'est associé à aucune donnée. Si le curseur est positionné sur un champ de ce type et que l'utilisateur presse (ENTER), le programme de gestion d'écran rend le contrôle au programme principal et passe une commande de sortie spéciale, *EsEmboite*. C'est alors au programme utilisateur d'appeler le deuxième écran de saisie. Voir le programme SAISIE.PAS sur la façon d'implémenter les écrans de saisie imbriqués.

ESE est l'écran auquel est ajouté le champ. *Mess* est le message à afficher à gauche (habituellement) du champ à modifier. *PLigne* et *PCol* sont les coordonnées écran pour le message. *Format* est le format de saisie facultatif du champ. Par défaut le masque de saisie sera composé entièrement de 'x' et aura une longueur égale à *LongEdit*. S'il n'est pas vide le format de saisie doit contenir au moins un caractère de formatage ('x', '!', 'A', etc.) pour réserver la position du curseur lorsqu'il sera sur ce champ. *eLigne* et *eCol* sont les coordonnées écran du champ. *LongEdit* est la largeur de ce champ. *LongEdit* est facultatif sauf si Format est vide. IndexAide est l'index à passer à la routine d'aide utilisateur si elle existe lorsque (F1) est pressée.

Exemple

```
AjoutChampEmboite(ESE, 'Précisions sur l''employé', 1, 1, '', 1,
                  14, 1, 1);
```

Dans cet exemple, *Employe* est un champ qui conduit à un deuxième écran de saisie contenant les champs relatifs aux différentes situations d'un employé (ancien employé, lieu d'emploi, nombre d'années de travail).

AjoutChampExt

Déclaration

```
procedure AjoutChampExt(var ESE : EsEnreg; Mess : string;
                        pLigne, pCol : byte;
                        Format : string; eLigne, eCol : byte;
                        IndexAide : Word;
                        ExtInf, ExtSup : Extended; Positions : byte;
                        var EditExt : Extended);
```

Objet

Ajoute un champ de type Extended à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type Extended à l'écran spécifié. Ces paramètres sont identiques à ceux de *AjoutChampBcd* sauf pour trois des quatre derniers paramètres qui sont de type Extended plutôt que de type BCD. Pour les explications sur les autres paramètres et des exemples appropriés voir *AjoutChampBcd*.

Remarque : Afin d'utiliser *AjoutChampExt*, vous devrez compiler TPSAISIE.PAS avec la directive de compilation {*N+}.

Voir aussi

AjoutChampBcd (TPSAISIE)

AjoutChampHeure

Déclaration

```
procedure AjoutChampHeure(var ESE : EsEnreg;
                          Mess : string;
                          pLigne, pCol : byte;
                          Format : ChaineDate;
                          eLigne, eCol, : byte;
                          IndexAide : Word;
                          HeureInf, HeureSup : Heure;
                          var EditHeure : Heure);
```

Objet

Ajoute un champ de type Heure à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type Heure à l'écran spécifié. Ses paramètres sont identiques à ceux des routines qui ajoutent des champs de type entier.

ESE est l'écran sur lequel est ajouté le champ Heure, *Mess* est le message à afficher à gauche (habituellement) du champ à modifier. *PLigne* et *PCol* sont les coordonnées écran du message. *Format* est le format de saisie du champ. Remarquez que ce paramètre est obligatoire pour les champs Heure. *eLigne* et *eCol* sont les coordonnées écran du champ à modifier. *IndexAide* est l'index à passer à une routine d'aide utilisateur si elle existe lorsque F1 est pressé. *HeureInf* et *HeureSup* représente les valeurs minimum et maximum de l'heure à modifier. Si les valeurs sont égales, il n'y a pas de contrôle d'intervalle. *EditHeure* est la variable Heure à modifier.

Voir le chapitre 3, partie C pour plus d'informations sur les variables Heure.

Exemple

```
HeureDebut : Heure;
...
AjoutChampHeure(ESE, 'Heure', 1, 1, 'hh:mm:ss te', 1, 6, 1, HeureMin,
                HeureMax, HeureDebut);
```

Ajoute un champ de type heure affiché dans le coin supérieur gauche de l'écran. Remarquez que l'heure est affichée et entrée en format am/pm.

AjoutChampInt

Déclaration

```
procedure AjoutChampInt(var ESE : EsEnreg; Mess : string;
                        pLigne, pCol : byte;
                        Format : string;
                        eLigne, eCol : byte;
                        IndexAide : Word;
                        IntInf, IntSup : Integer;
                        var EditInt : Integer);
```

Objet

Ajoute un champ de type entier à l'écran de saisie.

Commentaires

Cette routine ajoute un champ entier à l'écran spécifié. Ces paramètres sont identiques à ceux de *AjoutChampByte* mais les trois derniers paramètres sont de type entier plutôt que de type byte. Pour les explications sur les autres paramètres voir *AjoutChampByte*.

Voir aussi

AjoutChampByte (TPSAISIE)

AjoutChampLong

Déclaration

```
procedure AjoutChampLong(var ESE : EsEnreg;
                         Mess : string;
                         PLigne; PCol : byte;
                         Format : string;
                         eLigne, eCol : byte;
                         IndexAide : Word;
                         LongInf, LongSup : LongInt;
                         var EditLong : LongInt);
```

Objet

Ajoute un champ de type LongInt à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type LongInt à l'écran spécifié. Ses paramètres sont identiques à ceux de *AjoutChampByte* mais les trois derniers paramètres sont de type LongInt plutôt que de type byte. Voir les explications des autres paramètres dans *AjoutChampByte*.

Voir aussi

AjoutChampByte (TPSAISIE)

AjoutChampOuiNon

Déclaration

```
procedure AjoutChampOuiNon(var ESE : EsEnreg; Mess : string;
                           pLigne, pCol : byte; Format : string;
                           eLigne, eCol : byte; IndexAide : Word;
                           var AjoutChampOuiNon : Boolean);
```

Objet

Ajoute un champ de type Oui/Non à un écran de saisie.

Commentaires

Cette routine ajoute un champ Oui/Non à l'écran spécifié. Ces paramètres sont identiques à ceux de *AjoutChampBoolean*. Pour les explications sur les autres paramètres voir *AjoutChampBoolean*. La seule différence entre les deux procédures est qu'un masque de saisie égal à 'O' est pris en compte si *Format* est vide.

Voir aussi

AjoutChampBoolean (TPSAISIE)

AjoutChampReal

Déclaration

```
procedure AjoutChampReal(var ESE : EsEnreg; Mess : string;
                         pLigne, pCol : byte; Format : string;
                         eLigne, eCol : byte;
                         IndexAide : Word; RealInf, RealSup : Real;
                         Positions : byte; var EditReal : Real);
```

Objet

Ajoute un champ de type Real à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type Real à l'écran spécifié. Ces paramètres sont identiques à ceux de *AjoutChampBcd* mais trois des quatre derniers paramètres sont des réels sur 6 octets plutôt que des réels BCD. Pour les explications sur les autres paramètres et des exemples appropriés voir *AjoutChampBcd*.

Voir aussi

AjoutChampBcd (TPSAISIE)

AjoutChampSgl

Déclaration

```
procedure AjoutChampSgl(var ESE : EsEnreg; Mess : string;
                        pLigne, pCol : byte; Format : string;
                        eLigne, eCol : byte; IndexAide : Word;
                        SglInf, SglSup : Single; Positions : byte;
                        var EditSgl : Single);
```

Objet

Ajoute un champ de type Single à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type Single à l'écran spécifié. Ces paramètres sont identiques à ceux de *AjoutChampBcd* mais trois des quatre derniers paramètres sont de type Single plutôt que de type BCD. Pour les explications sur les autres paramètres et des exemples appropriés voir *AjoutChampBcd*.

Remarque : Afin d'utiliser *AjoutChampSgl* vous devrez compiler TPSAISIE.PAS avec la directive de compilation {$N+}.

Voir aussi

AjoutChampBcd (TPSAISIE)

AjoutChampShort

Déclaration

```
procedure AjoutChampShort(var ESE : EsEnreg; Mess : string;
                          PLigne; PCol : byte; Format : string;
                          eLigne, eCol, : byte; IndexAide : Word;
                          ShortInf, ShortSup : ShortInt;
                          var EditShort : ShortInt);
```

Objet

Ajoute un champ de type ShortInt à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type ShortInt à l'écran spécifié. Ses paramètres sont identiques à ceux de *AjoutChampByte* mais les trois derniers paramètres sont de type ShortInt plutôt que de type byte. Pour une explication sur les autres paramètres, voir *AjoutChampByte*.

Voir aussi

AjoutChampByte (TPSAISIE)

AjoutChampString

Déclaration

```
procedure AjoutChampString(var ESE : EsEnreg;
                           Mess : string;
                           pLigne, pCol : byte;
                           Format : string;
                           eLigne, eCol, LongEdit : byte;
                           IndexAide : Word;
                           Validation : Pointer;
                           var EditeChn : String);
```

Objet

Ajoute un champ de type String à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type String à l'écran spécifié. Les principales différences entre cette routine et les autres routines d'ajout de champs sont :

- elle comporte un paramètre qui est un pointeur vers une routine de contrôle spéciale
- elle fournit le moyen de traiter des chaînes plus longues que le champ à modifier.

ESE est l'écran sur lequel est ajouté le champ string, *Mess* est le message à afficher à gauche (habituellement) du champ à modifier. *PLigne* et *PCol* sont les coordonnées écran du message. *Format* est le format de saisie facultatif pour le champ. Par défaut, un champ de saisie de longueur *LongEdit* composé entièrement de 'x' sera pris en compte (il permet la saisie de n'importe quel caractère). *eLigne* et *eCol* sont les coordonnées écran du champ à modifier. *LongEdit* est la largeur du champ à modifier.

Si *LongEdit* < *Length(Format)*, le champ sera déroulé horizontalement pendant la modification. *IndexAide* est l'index à passer à une routine d'aide utilisateur si elle existe lorsque (F1) est pressé. Si ce paramètre est différent de nil, la routine de validation doit pointer sur une routine interne ou une routine écrite par l'utilisateur. *EditeChn* est la chaîne à modifier, elle doit avoir comme longueur le maximum de *LongEdit* et *Length(Format)*.

Exemple

```
AjoutChampString(ESE, 'Remarques', 19, 18, ChaineCarac('X',254), 19,
                 25, 46,16, Nil, Fiche.Remarques);
```

Ici, *Fiche.Remarques* est de type String[254], ainsi qu'on peut le voir dans le masque de format. *LongEdit* étant égal à 46, le champ devra se dérouler horizontalement à l'édition si la chaîne dépasse 46 caractères.

```
AjoutChampString(ESE, 'Département', 10, 18, '##', 10, 25, 2, 4,
                 @BlancsInterdits, Fiche.Departement);
```

Fiche.Departement est de type String[2], comme l'indiquent le masque de format et le paramètre *LongEdit*. *BlancsInterdits* est une routine de validation de *Turbo Professionnel* qui sera abordée plus loin dans ce chapitre.

Voir aussi

AjoutChampArray (TPSAISIE)

BlancsInterdits (TPSAISIE)

AjoutChampUtl

Déclaration

```
procedure AjoutChampUtl(var ESE : EsEnreg; Mess : string;
                        pLigne, pCol : byte; Format : string;
                        eLigne, eCol, LongEdit : byte; IndexAide : Word;
                        Validation, Conversion : Pointer;
                        PtrUtl1, PtrUtl2 : Pointer; var EditVar);
```

Objet

Ajoute un champ de type défini par l'utilisateur à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type défini par l'utilisateur à l'écran spécifié. Elle est destinée aux programmeurs avertis qui ont écrit leurs propres routines de validation et de conversion pour prendre en compte des types de données non supportés par TPSAISIE.

ESE est l'écran de saisie où le champ est ajouté, *Mess* est le message à afficher à gauche (habituellement) du champ à modifier. *PLigne* et *PCol* sont les coordonnées écran du message. *Format* est le format de saisie facultatif du champ. Par défaut un masque de saisie de longueur *LongEdit* composé entièrement de 'x' est pris en compte (permet d'entrer n'importe quel caractère). *eLigne* et *eCol* sont les coordonnées écran du champ à modifier. *LongEdit* est la largeur du champ à modifier ; si *LongEdit* < *Length(Format)* le champ sera déroulé horizontalement si pendant l'édition. *IndexAide* est l'index à passer à une routine d'aide utilisateur si elle existe lorsque (F1) est pressé. *Validation* doit pointer sur une routine de contrôle interne ou écrite par l'utilisateur et *Conversion* doit pointer sur une routine de conversion utilisateur. Les deux variables sont obligatoires pour les champs définis par l'utilisateur. *PtrUtl1* et *PtrUtl2* peuvent être utilisés pour enregistrer des pointeurs sur les données utilisées par les routines de conversion et de contrôle. *EditVar* est la variable à modifier.

Voir "Types définis par l'utilisateur" au début de ce chapitre pour plus de détails.

Exemple

```
MaVar : MonTypeDeDonnee;
...
AjoutChampUtl(ESE, 'Quelque chose', 1, 1, '', 1, 11, 20, 1,
     @MaRoutineDeValidation, @MaRoutineDeConversion, Nil, Nil, MaVar);
```

Cet exemple ajoute un champ de type *MonTypeDeDonnee* à l'écran d'édition. *MaRoutineDeValidation* et *MaRoutineDeConversion* sont des routines écrites par l'utilisateur dont le but est de gérer la validation et la conversion des champs de type *MonTypeDeDonnee*.

AjoutChampWord

Déclaration

```
procedure AjoutChampWord(var ESE : EsEnreg; Mess : string;
                         pLigne, pCol : byte; Format : string;
                         eLigne, eCol : byte; IndexAide : Word;
                         WordInf, WordSup : Word; var EditWord : Word);
```

Objet

Ajoute un champ de type Word à un écran de saisie.

Commentaires

Cette routine ajoute un champ de type Word à l'écran spécifié. Ces paramètres sont identiques à ceux de *AjoutChampByte* mais les trois derniers paramètres sont de type Word plutôt que de type byte. Voir les explications sur les autres paramètres dans *AjoutChampByte*.

Voir aussi

AjoutChampByte (TPSAISIE)

BlancsInterdits

Déclaration

```
procedure BlancsInterdits(var Ec : EnrChamp;
                          var CodeErr : Byte;
                          var ChnErreur : PrtChaine) : Boolean;
```

Objet

Vérifie que les sous-champs utiles d'une chaîne ne contiennent pas de blancs.

Commentaires

Cette routine de contrôle vérifie qu'un champ est rempli. Comme toutes les routines de contrôle de haut niveau, elle ne devra jamais être appelée répétitivement. Elle ne devra être référencée seulement que par son adresse. Remarquez que l'utilisation de cette routine de contrôle impose des contraintes plus rigoureuses qu'un champ obligatoire puisque les champs obligatoires peuvent être partiellement remplis.

Exemple

```
AjoutChampString(ESE, 'Département', 10, 8, '##', 10, 25, 2, 4,
                 @BlancsInterdits, Fiche.Departement);
```

En spécifiant *BlancsInterdit* comme routine de contrôle pour ce champ, nous garantissons à la fois que l'utilisateur ne laissera pas le champ vide et qu'il ne pourra pas entrer une seule lettre dans l'état.

Voir aussi

InitObligatoire (TPSAISIE)

ChangeProtection

Déclaration

```
procedure ChangeProtection(var ESE : EsEnreg; IDChamp : byte;
                           OnOff : boolean);
```

Objet

Modifie le statut de protection d'un champ.

Commentaires

Habituellement un champ reçoit son mode de protection au moment où il est ajouté à un écran de saisie et il garde son statut de protection aussi longtemps que l'écran existe. Dans certains cas, quoi qu'il en soit, vous pourrez être amené à protéger ou déprotéger un champ sur certaines réponses de l'utilisateur. Par exemple, vous pourrez protéger un champ dès que l'utilisateur aura saisi une donnée valide ou vous pourrez déprotéger un champ qui avait été initialement protégé lorsque l'utilisateur aura saisi une donnée valide dans un autre champ. *ChangeProtection* vous fournit les moyens de le faire.

Néanmoins dans certaines situations, prenez des précautions. Si le mode courant d'enroulement est *ArretAuxBords*, vous devrez faire très attention en utilisant *ChangeProtection* sur des champs situés près des bords de l'écran si la procédure *EcranEdit* est encore active. En effet, lorsque *EcranEdit* est appelé en premier et que le mode d'enroulement courant est *ArretAuxBords*, *EcranEdit* balaie l'écran de saisie pour trouver le premier et le dernier champ non protégé de l'écran, modifie les options d'avance automatique pour les deux champs et désactive effectivement certaines commandes de déplacement du curseur lorsque l'un ou l'autre des champs est actif. En conséquence, il n'est pas recommandé de modifier le statut de protection d'un champ lorsque le mode d'enroulement est *ArretAuxBords* sauf si vous avez la garantie que le programme de gestion de l'écran ne sera pas perturbé.

Exemple

```
ChangeProtection(ESR, 5, On);
```

Le sixième champ de l'écran de saisie désigné est mis en mode protégé (les numéros de champs commencent à zéro).

Voir aussi

InitProtection (TPSAISIE) InitModeEnroule (TPSAISIE)

DesactiveSaisieSouris

Déclaration

```
procedure DesactiveSaisieSouris;
```

Objet

Désactive la souris dans TPSAISIE.

Commentaires

Cette routine désactive le support de la souris dans TPSAISIE. Outre l'effacement de l'indicateur interne initialisé par *ActiveSaisieSouris,* elle désactive le gestionnaire d'événements de la souris de TPSOURIS et initialise *PtrToucheSaisie* pour pointer sur *LitMotTouche* (TPCRT).

Voir aussi

ActiveSaisieSouris (TPSAISIE)

DessineChamp

Déclaration

```
procedure DessineChamp(var ESE : EsEnreg; IDChamp : byte);
```

Objet

Affiche le champ correspondant à *IDChamp.*

Commentaires

Cette routine ré-affiche le champ correspondant à *IDChamp* sur l'écran de saisie spécifié. L'utilisation la plus fréquente est une routine de post-édition qui calcule des champs qui doivent être ré-affichés lorsque des nouvelles valeurs ont été calculées. Voir le programme SAISIE.PAS comme exemple d'utilisation de *DessineChamp.*

Exemple

```
DessineChamp(ESE, ESE.IDCourant);
```

Ré-affiche le champ courant sur l'écran de saisie spécifié.

Voir aussi

DessineEcranEdit (TPSAISIE) InitPtrApresEdit (TPSAISIE)

DessineEcranEdit

Déclaration

```
procedure DessineEcranEdit(var ESE : EsEnreg);
```

Objet

Redessine complètement l'écran de saisie.

Commentaires

Cette routine appelle *DessineChamp* de manière répétitive afin d'afficher chaque champ de l'écran de saisie. Puisque *EcranEdit* le fait automatiquement lorsque l'édition commence, l'utilisation de *DessineEcranEdit* est rarement nécessaire. Cependant, elle est particulièrement utile lorsque vous voulez afficher des données sans autoriser leur modification. Voir la procédure *DisplayRecord* dans le programme de démonstration GENETAT.PAS pour un exemple d'utilisation de *DessineEcranEdit*.

Voir aussi

DessineChamp (TPSAISIE)

EcranEdit

Déclaration

```
function EcranEdit(var ESE : EsEnreg; ChampDebut : word ;
                   LectSeulement : Boolean) : TypeEs;
```

Objet

Edition des champs jusqu'à ce qu'une commande de sortie soit exécutée.

Commentaires

Cette routine est la routine principale de TPSAISIE, elle contient le coeur du programme de gestion des écrans de saisie. Elle est en général appelée dans une boucle *repeat..until*, dans laquelle le programme de gestion de l'écran et les éditeurs de champs gèrent toutes les commandes d'édition et de déplacement du curseur alors que le programme utilisateur gère les commandes de sortie renvoyées en résultat de la fonction *EcranEdit*.

Dans la plupart des cas, le programme utilisateur n'est concerné que par quatre commandes de sortie : *EsEnrSuiv* qui indique que l'utilisateur souhaite éditer l'enregistrement suivant dans une suite d'enregistrements ; *EsEnrPreced* qui indique qu'il souhaite éditer l'enregistrement précédent, *EsFinSaisie* qui indique que l'édition est terminée et *EsQuitte* qui indique qu'il désire s'arrêter. Remarquez que *EsEnrSuiv* et *EsEnrPreced* n'ont de sens que dans les applications type base de données pour lesquels il y a plusieurs enregistrements à éditer. Les autres commandes de sortie valides sont les commandes de sortie définies par l'utilisateur, *EsUtl0..EsUtl9*. Au cas où l'écran de saisie n'aurait pas été initialisé correctement ou contiendrait des champs non déprotégés, la fonction pourrait renvoyer *EsRien*.

Le paramètre *ChampDebut* indique la position initiale du curseur à l'intérieur de l'écran de saisie. Les valeurs les plus fréquentes sont 0, affectée au premier champ de l'écran, et *ESE.IDCourant* (voir les exemples ci-dessous). Si le champ spécifié est un champ protégé, le programme de gestion de l'écran placera le curseur sur le premier champ non protégé suivant le champ spécifié. Le paramètre *LectSeulement* est utilisé pour sélectionner le mode "Lecture seulement" avec lequel l'utilisateur peut se déplacer de champ en champ et d'enregistrement en enregistrement mais sans pouvoir effectuer de modifications.

Pour un exemple d'utilisation de *EcranEdit* aussi bien que d'implémentation des commandes de sortie définies par l'utilisateur, voir le programme SAISIE.PAS.

Exemple

```
CommandeSortie := EcranEdit(ESE, 0, False);
```

Commence une session d'édition et place le curseur sur le premier champ (non protégé) de l'écran de saisie.

```
CommandeSortie := EcranEdit(ESE, ESE.IDCourant, True);
```

Commence (ou poursuit) une session d'édition avec le curseur placé sur le champ qu'il occupait au moment ou *EcranEdit* a été appelé. Pour en observer les effets, chargez le programme de démonstration SAISIE.EXE, pressez (CTRL)-(PGDN) puis (PGUP). Le troisième paramètre sélectionne le mode lecture-seulement.

EditeChaine

Déclaration

```
procedure EditeChaine(Mess : string;
                      Ligne, Col : byte;
                      LongEdit : byte;
                      Format : string;
                      IndexAide : Word;
                      var Echappement : boolean;
                      var EditeChn : string);
```

Objet

Utilise l'un des éditeurs de champ pour modifier une chaîne de caractère.

Commentaires

Cette routine permet d'utiliser les deux éditeurs champs (l'éditeur chaîne et l'éditeur numérique) comme des éditeurs lignes bien mieux que la procédure *LitChaine* de TPEDIT. Cette routine vous permet de spécifier un masque de saisie.

Pour d'autres raisons, cependant, il sera parfois préférable d'appeler *EditeChaine* plutôt que *AjoutChampString* : toutes les options spéciales d'édition doivent être sélectionnées en appelant des routines appropriées *InitOption* préalablement. Par exemple, pour initialiser les couleurs du message affiché, vous devrez appeler *InitAttrMess* et pour utiliser l'éditeur numérique, vous devrez appeler *InitNumerique*. (Remarquez que si *Format* contient '$' ou ',', le mode numérique est sélectionné automatiquement sauf si *IncludeNumeric* n'est pas défini, auquel cas le programme s'arrêtera).

Une autre différence entre *EditeChaine* et *LitChaine* est que la chaîne à modifier sera complétée sur toute sa longueur avec des blancs sauf a) si l'option de suppression de blancs est activée, elle l'est par défaut, ou b) l'éditeur numérique est utilisé. En mode numérique les caractères spéciaux de format (tels que '$' ou ',') sont également éliminés de la chaîne avant de la renvoyer. Si vous utilisez *EditeChaine* pour modifier des nombres, vous devez par conséquent convertir le nombre en chaîne de caractères préalablement et le convertir en nombre au retour. La chaîne numérique passée initialement (*EditeChn*) ne devra pas contenir de caractères spéciaux mais simplement un nombre brut tel que '-10000.12' ou une chaîne vide. Si l'éditeur chaîne est utilisé, *EditeChn* pourrait être vide ou préformatée. Par exemple, si le masque de saisie est '(999)999-9999', *EditeChn* pourrait contenir '( ) _ '.

Les paramètres de *EditChaine* sont en général semblables à ceux de *AjoutChampString*. *Mess* est le message affiché à gauche du champ à modifier. *Ligne* et *Col* sont les coordonnées écran du message. *Format* est un format de saisie facultatif pour le champ. Si aucun masque n'est spécifié, celui-ci aura la longueur de *LongEdit* et contiendra exclusivement des 'x' (permet la saisie de n'importe quel caractère). *LongEdit* est la largeur du champ à modifier qui commence après la fin de *Mess*. Si *LongEdit* < *Length(Format)*, le champ sera déroulé horizontalement pendant l'édition. *IndexAide* est l'index à passer à la routine d'aide définie par l'utilisateur si elle existe, lorsque (F1) est pressée. *Echappement* sera initialisé à "True" à la sortie si la commande *EsQuitte* (normalement affectée à (ESC)) est exécutée. *EditeChn* est la chaîne à modifier, elle a comme longueur la taille maximum de *LongEdit* et *Length(Format)*.

Exemple

```
NomFic := '';
EditeChaine('Nom du fichier: ', 1, 1, 64, '', 0, Escape, NomFic);
```

Utilise l'éditeur chaîne pour obtenir un nom de fichier.

```
Nombre := '';
InitNumerique(On);
EditeChaine('Entrez un nombre: ', 1, 1, 5, '99999', 0, Escape,
            Nombre);
InitNumerique(Off);
```

Utilise l'éditeur numérique pour obtenir un nombre compris dans l'intervalle 0-99 999.

Voir aussi

AjoutChampString (TPSAISIE) LitChaine (TPEDIT)

EffaceEntFen

Déclaration

```
procedure EffaceEntFen(var ESE : EsEnreg);
```

Objet

Efface la fenêtre de l'écran de saisie.

Commentaires

Efface la fenêtre de saisie associée à l'écran spécifié. Si aucune coordonnée de fenêtre n'a été mentionnée (avec *InitFenEnt*), l'écran tout entier sera effacé.

Exemple

```
InitFenEnt(ESE, 1, 5, 80, 20, False, TextAttr, TextAttr);
...
EffaceEntFen(ESE);
```

Efface les colonnes 1-80 de l'écran à partir de la ligne 5 jusqu'à la ligne 20.

Voir aussi

InitFenEnt (TPSAISIE)

InitAttrChaine

Déclaration

```
procedure InitAttrChaine(A : Byte);
```

Objet

Initialise l'attribut de chaîne.

Commentaires

Cette routine vous permet d'initialiser l'attribut (appelé attribut chaîne) utilisé pour afficher les champs lorsqu'ils sont sélectionnés pour modification. La valeur passée à *InitAttrChaine* est enregistrée dans la variable globale *EsAttrChaine*.

Exemple

```
InitAttrChaine($70);
```

Affiche les chaînes à modifier en inversion vidéo.

Voir aussi

InitAttrCtrl (TPSAISIE)
InitAttrChamp (TPSAISIE)
InitAttrMess (TPSAISIE)

InitAttrChamp

Déclaration

```
procedure InitAttrChamp(A : Byte);
```

Objet

Initialise l'attribut du champ.

Commentaires

Cette routine vous permet d'initialiser l'attribut (appelé l'attribut champ) utilisé pour afficher les champs lorsqu'ils ne sont pas sélectionnés pour modification. La valeur passée à *InitAttrChamp* est enregistrée dans la variable globale *EsAttrChamp*.

Exemple

```
InitAttrChaine($70);
InitAttrChamp($0F);
```

Affiche des chaînes à modifier en inversion vidéo puis affiche les champs inactifs en blanc sur noir.

Voir aussi

InitAttrCtrl (TPSAISIE)
InitAttrChaine (TPSAISIE)
InitAttrMess (TPSAISIE)

InitAttrCtrl

Déclaration

```
procedure InitAttrCtrl(A : Byte);
```

Objet

Initialise l'attribut pour les caractères de contrôle.

Commentaires

Cette routine vous permet d'initialiser l'attribut utilisé pour afficher les caractères de contrôle imbriqués dans des chaînes à modifier. Si cet attribut est le même que l'attribut de la chaîne (initialisé avec *InitAttrChaine*), les caractères de contrôle ne pourront être saisis. S'il est différent, les caractères de contrôle peuvent être saisis en utilisant la commande (CTRL)-P (*EsCaracCtrl*) et ils seront affichés comme des caractères majuscules (^U comme 'U'), avec l'attribut désigné. La valeur passée à *InitAttrCtrl* est enregistrée dans la variable globale *EsAttrCtrl*.

Exemple

```
InitAttrChaine($7);
InitAttrCtrl($F);
```

Affiche les chaînes à modifier en gris clair sur noir et les caractères de contrôle en blanc sur noir.

Voir aussi

InitAttrChamp (TPSAISIE)
InitAttrChaine (TPSAISIE)
InitAttrMess (TPSAISIE)

InitAttrMess

Déclaration

```
procedure InitAttrMess(A : Byte);
```

Objet

Initialise l'attribut du message.

Commentaires

Cette procédure vous permet d'initialiser l'attribut (appelé l'attribut de message), utilisé pour afficher le message associé à un champ donné. La valeur passée à *InitAttrMess* est enregistrée dans la variable globale *EsAttrMess*.

Exemple

```
InitAttrChaine($70);
InitAttrMess($07)
```

Affiche les chaînes à modifier en inversion vidéo et le message en gris clair sur noir.

Voir aussi

InitAttrCtrl (TPSAISIE)
InitAttrChaine (TPSAISIE)
InitAttrChamp (TPSAISIE)

InitAutoNumLock

Déclaration

```
procedure InitAutoNumLock(On : Boolean);
```

Objet

Active ou désactive l'option de (NUM LOCK) automatique.

Commentaires

Cette procédure vous permet d'activer une option qui active automatiquement (NUM LOCK) chaque fois qu'un champ particulier est modifié et qui la restaure à son état précédent ensuite.

Exemple

```
InitAutoNumLock(On);
AjoutChampReal(ESE, 'Salaire hedbomadaire', 16, 12, '999,99 F', 16,
               25, 12, 0, 999.99, 0, Fiche.Salaire);
InitAutoNumLock(Off);
```

Dans ce cas, le (NUM LOCK) automatique est activé pour le champ *Fiche.Salaire* et désactivé pour tous les autres champs.

Voir aussi

InitNumerique (TPSAISIE)

InitAvanceAuto

Déclaration

```
procedure InitAvanceAuto(on : Boolean);
```

Objet

Active ou désactive l'option d'avance automatique.

Commentaires

L'état de l'option d'avance automatique pour un champ donné défini ce qui arrivera dans plusieurs situations courantes. Par exemple : 1) Le curseur est au début d'un champ et l'utilisateur presse (←) : si l'option d'avance automatique est positionnée, comme elle l'est par défaut, le curseur se placera sur le champ à gauche du champ courant. Si cette option est désactivée, rien ne se produira. 2) Le champ courant qui contient un maximum de cinq caractères contient maintenant quatre caractères et l'utilisateur vient juste d'entrer le cinquième : si l'option d'avance automatique est activée, le curseur avancera sur le prochain champ, dans le cas contraire l'utilisateur pourra presser (ENTER) pour l'avancer. 3) Le curseur est à la fin du champ courant et l'utilisateur presse (CTRL)-(→) pour se placer sur le prochain mot à droite : si l'option d'avance automatique est activée, le curseur se placera au début du premier mot dans le champ suivant situé à droite; dans le cas contraire, rien ne se produira.

Exemple

```
InitAvanceAuto(Off);
AjoutChampString(ESE, 'Remarques', 19, 18, ChaineCarac('X', 254), 19,
                 25, 46, 16, Nil, Fiche.Remarques);
InitAvanceAuto(On);
```

Désactive l'option d'avance automatique pour le champ *Fiche.Remarques* (la chaîne peut être plus longue que le champ à modifier et de ce fait, il est difficile de savoir où commence et où se termine la chaîne).

InitBipSurErreur

Déclaration

```
procedure InitBipSurErreur(var ESE : EsEnreg; On : Boolean);
```

Objet

Emet un bip lorsqu'un caractère non valide est saisi.

Commentaires

Habituellement lorsque l'utilisateur essaie d'entrer un caractère non valide dans un champ (par rapport au masque de saisie de ce champ) ou une touche telle que (F5) qui n'a pas de rôle particulier, l'éditeur de champ ignore la touche. En appelant *InitBitSurErreur*, cependant, vous pouvez demander qu'un bip bref soit émis pour informer l'utilisateur de l'erreur. Vous pouvez contrôler le son qui est émis en appelant la procédure *InitCloche*. Voir le programme SAISIE.PAS pour un exemple d'utilisation de *InitBipSurErreur*.

Exemple

```
InitBipSurErreur(ESE, On);
```

Emet un bip lorsque l'utilisateur essaie d'entrer un caractère non valide.

Voir aussi

InitCloche (TPSAISIE)

InitCaracComplement

Déclaration

```
procedure InitCaracComplement(Ch : Char);
```

Objet

Positionne le caractère utilisé pour compléter une chaîne.

Commentaires

Cette routine vous permet de spécifier le caractère utilisé pour compléter une chaîne sur toute sa longueur. Le caractère par défaut est l'espace (' '). La chaîne n'est complétée que d'un seul côté. Si le champ est justifié à gauche, comme le sont la plupart des champs, la chaîne sera complétée à droite. Si le champ est justifié à droite, elle sera complétée à gauche. Voir le programme SAISIE.PAS pour quelques exemples d'utilisation de *InitCaracComplement*.

Exemple

```
InitCaracComplement('*');
AjoutChampReal(ESE, 'Hebdomadaire', 16, 49, '####,## F', 16, 57, 13,
               0 ,0, 0, Fiche.Hebdomadaire);
InitCaracComplement(' ');
```

Initialise le caractère de complément à astérisque, ajoute un champ puis réinitialise le caractère de complément à espace.

Voir aussi

InitJustifDroite (TPSAISIE)

InitCaracMotDePasse

Déclaration

```
procedure InitCaracMotDePasse(Ch : Char);
```

Objet

Initialise le caractère utilisé en mode mot de passe.

Commentaires

Cette routine vous permet de spécifier le caractère utilisé pour représenter le texte en mode mot de passe. La valeur par défaut est ^G (un tiret au milieu du caractère). Remarquez que cette procédure a des effets globaux. Vous ne pouvez pas utiliser un caractère mot de passe différent pour des champs différents dans un écran de saisie.

Exemple

```
InitCaracMotDePasse('*');
InitModeMotDePasse(On);
EditeChaine('Mot de passe : ', 1, 1, 10, '', 0, Escape, MotPasse);
InitModeMotDePasse(Off);
```

Tous les caractères entrés par l'utilisateur seront affichés comme des astérisques.

InitCloche

Déclaration

```
procedure InitCloche(Frequence, Duree, : Word);
```

Objet

Initialise la fréquence et la durée d'un son.

Commentaires

Cette procédure vous permet de contrôler le son généré par des appels à *SonneCloche*. Les valeurs passées à *SonneCloche* sont enregistrées dans les variables globales *FrequenceCloche* et *DureeCloche*. Leurs valeurs par défaut sont 700 (hertz) et 100 (millisecondes) respectivement.

Exemple

```
InitCloche(500, 250);
```

Initialise un son d'une fréquence de 500 hertz et d'une durée de 1/4 de seconde.

Voir aussi

SonneCloche (TPSAISIE)

InitCurseurEnFin

Déclaration

```
procedure InitCurseurEnFin(On : Boolean);
```

Objet

Active ou désactive l'option de positionnement de curseur en fin de champ.

Commentaires

Habituellement, lorsque le curseur est placé sur le champ qui contient déjà un texte, il est positionné au début de ce champ. *InitCurseurEnFin* vous permet de positionner le curseur à la fin du champ pour pouvoir le compléter.

Exemple

```
InitCurseurEnFin(On);
AjoutChampString(ESE, 'Nom', 7, 19, '', 7, 25, 30, 1, nil, Fiche.Nom);
InitCurseurEnFin(Off);
```

Si le champ *Fiche.Nom* contient déjà un texte, le curseur sera positionné à la fin de la chaîne, lorsqu'il est placé dans ce champ.

Voir aussi

InitEffPremCar (TPSAISIE)

InitDelimiteurs

Déclaration

```
procedure InitDelimiteurs(var : ESE : EsEnreg;
                          gauche, droite : Char;
                          AttrChamp, AttrSelect : Byte);
```

Objet

Positionne les délimiteurs de champs pour l'écran de saisie.

Commentaires

Cette option vous permet de spécifier un couple de caractères qui seront affichés immédiatement à gauche et à droite de chaque champ dans l'écran de saisie désigné. Les paramètres gauche et droite indiquent quels caractères doivent être utilisés comme délimiteurs. *AttrChamp* spécifie les attributs d'affichage de ces délimiteurs. *AttrSelect* spécifie les attributs avec lesquels ils seront affichés lorsque le champ sera sélectionné pour modification. Par défaut les délimiteurs de champ pour un écran de saisie sont égaux à nul (#0).

Exemple

```
InitDelimiteurs(ESE, '[', ']', $0F, $70);
```

Tous les champs seront délimités par des crochets et les crochets seront affichés en blanc sur noir pour les champs inactifs et en inversion vidéo pour les champs sélectionnés.

InitEffPremCarac

Déclaration

```
procedure InitEffPremCarac(On : Boolean);
```

Objet

Active ou désactive l'option d'effacement du premier caractère.

Commentaires

Habituellement lorsque le curseur est placé dans un champ qui contient déjà un texte et que l'utilisateur commence à saisir des caractères, le nouveau texte est inséré (si le mode insertion est positionné) ou recouvre l'ancien texte (si le mode recouvrement est positionné). *InitEffPremCarac* vous permet de spécifier cela si le premier caractère entré est un caractère ASCII (contrairement à une commande d'édition), le champ sera effacé et le nouveau caractère inséré au début du champ.

Exemple

```
InitEffPremCarac(On);
AjoutChampString(ESE, 'Adresse', 8, 16, '', 8, 25, 30, 2, nil,
                     Fiche.Adresse);
InitEffPremCarac(Off);
```

Dans cet exemple, l'option d'effacement du premier caractère est positionnée pour le champ *Fiche.Adresse* et désactivée pour tous les autres champs.

InitElimineBlancs

Déclaration

```
procedure InitElimineBlancs(On : Boolean);
```

Objet

Active ou désactive la suppression de blancs.

Commentaires

Cette routine vous permet de déterminer si les blancs de tête ou de queue doivent être supprimés. Soit le masque de saisie '!!!!!' et la chaîne ' XYZ'. Si l'option de suppression de blancs est active, le résultat sera 'XYZ', sinon le résultat sera identique à la donnée saisie.

InitElimineBlancs devrait être appelée avant d'ajouter le premier champ.

Par défaut la suppression de blancs est active. Remarquez que la suppression de blancs est automatiquement désactivée pour les champs dont les masques de saisie contiennent des caractères littéraux ou semi-littéraux.

Exemple

```
InitElimineBlancs(Off);
AjoutChampString(ESE, 'Remarques', 19, 18, ChaineCarac('X', 254), 19,
                 25, 46, 16, nil, Fiche.Remarques);
InitElimineBlancs(On);
```

Puisque la suppression de blancs est désactivée, *Fiche.Remarques* sera toujours complété avec des blancs sur une longueur maximum de 254.

Voir aussi

AjoutChampArray (TPSAISIE)

InitEnregEs

Déclaration

```
procedure InitEnregEs(var ESE : EsEnreg);
```

Objet

Initialise un enregistrement de type *EsEnreg*.

Commentaires

Cette procédure doit être appelée avant d'ajouter un champ sur l'écran de saisie. Elle initialise toutes les variables de *EsEnreg* avec leurs valeurs par défaut et insère une signature qui permet aux autres routines de savoir si *EsEnreg* a été initialisé correctement ou non. Si ce n'est pas le cas, et que vous essayez d'y ajouter un champ, le programme s'arrêtera.

Exemple

```
InitPtrAvantEdit(ESE, @AfficheMessageAide);
InitEnregES(ESE);
```

Incorrect ! *ESE* doit être initialisée avant d'appeler *InitPtrAvantEdit*.

```
InitEnregES(ESE);
InitPtrAvantEdit(ESE, @AfficheMessageAide);
```

Correct.

InitFenEnt

Déclaration

```
procedure InitFenEnt(var ESE : EsEnreg;
                     Xbas, Ybas, Xhaut, Yhaut : Byte;
                     Encadre : Boolean; AttrBack, AttrCadre : Byte);
```

Objet

Initialise les coordonnées et la couleur de l'arrière-plan pour une fenêtre de saisie.

Commentaires

Cette routine vous permet de spécifier les coordonnées écran pour la fenêtre dans laquelle l'écran de saisie sera affiché. Le paramètre *AttrBack* spécifie l'attribut vidéo utilisé lorsque lors de l'effacement de la fenêtre (avec *EffaceEntFen*) et lors du déroulement de la fenêtre. Si cette routine n'est pas appelée, l'écran est égal à la fenêtre et la valeur de *TextAttr* au moment ou *InitEnregEs* a été appelée, est considéré comme l'attribut d'arrière-plan.

Encadre indique si la fenêtre possède ou non un cadre, si ce paramètre vaut "True", les caractères de cadre de *CaracsCadre(TPCRT)* seront utilisés pour dessiner le cadre. *AttrCadre* spécifie la couleur du cadre. Notez aussi que si *Encadre* vaut "True", les coordonnées de l'écran spécifié définissent l'emplacement de dessin du cadre et les coordonnées pour la fenêtre de saisie actuelle seront adaptées conformément.

Exemple

```
InitFenEnt(ESE, 1, 5, 80, 20, False, TextAttr, TextAttr);
```

Sélectionne les lignes 5 à 20 comme fenêtre dans laquelle l'écran de saisie associé à *ESE* sera affiché. La fenêtre ne comporte pas de cadre.

Voir aussi

EffaceEntFen (TPSAISIE)

InitEnregEs (TPSAISIE)

InitForceMode

Déclaration

```
procedure InitForceMode(Force, Recouvre : Boolean);
```

Objet

Force le mode insertion/recouvrement, sinon utilise le paramétrage précédent.

Commentaires

Habituellement, le mode insertion est le mode par défaut et lorsque l'utilisateur passe au mode recouvrement, les éditeurs resteront en mode recouvrement jusqu'à ce que l'utilisateur change de mode. Dans certains cas, quoi qu'il en soit, il peut être souhaitable de forcer le mode insertion ou recouvrement temporairement pour un champ donné. *InitForceMode* vous permet de demander qu'un mode particulier soit forcé chaque fois qu'un champ est modifié. L'utilisateur peut encore changer de mode, quoi qu'il en soit. Vous pouvez simplement contrôler l'état du mode lorsque l'édition commence.

Si le paramètre *Force* vaut "True" alors le paramètre *Recouvre* indique si le mode insertion ou recouvrement doit être forcé. Si le paramètre *Force* vaut "False", le paramètre *Recouvre* est ignoré et le mode insertion/recouvrement est géré normalement.

Exemple

```
InitForceMode(True, False);
AjoutChampReal(ESE, 'Salaire horaire', 16, 12, '999,99 F', 16, 25, 12,
               0, 999.99, 0, Fiche.Salaire);
InitForceMode(False, False);
```

Chaque fois que le champ *Fiche.Salaire* est modifié, le mode courant (insertion/recouvrement) sera sauvegardé et le mode insertion sera forcé. Une fois que le champ a été modifié, le mode sera restauré à sa valeur précédente.

InitInsereEtPousse

Déclaration

```
procedure InitInsereEtPousse(On : Boolean);
```

Objet

Active ou désactive l'option de décalage à droite du texte.

Commentaires

Par défaut tout texte entré dans un sous-champ en mode insertion poussera vers la droite le texte existant et tout caractère situé à l'extrémité droite du sous-champ sera perdu. Vous pouvez changer cela en exécutant "*InitInsereEtPousse(Off)*". Lorsque l'option de décalage à droite est désactivée, les caractères de droite ne seront pas perdus et le texte pourra être inséré seulement si le dernier caractère du sous-champ courant contient un espace. Remarquez que ce paramétrage modifie seulement le comportement de l'éditeur chaîne. L'éditeur numérique n'autorise jamais l'insertion de texte lorsque le champ courant est plein.

Exemple

```
InitInsereEtPousse(Off);
AjoutChampString(ESE, 'Ville', 9, 19, '', 9, 25, 25, 3, nil,
                 Fiche.Ville);
InitInsereEtPousse(On);
```

Dans cet exemple, l'option d'insertion et de décalage à droite est désactivée pendant l'édition du champ *Fiche.Ville* et activée pendant l'édition des autres champs.

InitJustifDroite

Déclaration

```
procedure InitJustifDroite(On : Boolean);
```

Objet

Active ou désactive la justification droite.

Commentaires

Cette routine vous permet de justifier un champ à droite. Elle est utilisée principalement avec des champs contenant des nombres. Remarquez cependant que les champs numériques, ajoutés après une instruction *InitNumerique*(On) et les champs contenant des points décimaux fixes sont automatiquement justifiés à droite. Si le champ à justifier à droite n'est pas un nombre, il est prudent de sélectionner l'option de suppression de blancs pour ce champ.

N'oubliez pas que, sauf pour les champs numériques et les champs contenant un point décimal fixe, la justification à droite apparaît seulement au début et à la fin du champ. Ainsi, si la chaîne à modifier contient '123' et que le masque de saisie est '99999', le résultat sera ' 123'. Si l'utilisateur saisit alors '456', '456 ' sera affiché jusqu'à ce qu'il quitte le champ, alors qu'il sera ré-affiché comme ' 456'.

Exemple

```
InitJustifDroite(On);
AjoutChampByte(ESE, 'Age', 14, 52, '999', 14, 57, 10, 0, 0, Age);
InitJustifDroite(Off);
```

Active l'option de justification à droite, ajoute le champ *Age* puis désactive à nouveau l'option de justification à droite si bien que le nouveau champ sera justifié à gauche.

Voir aussi

InitCaracComplement (TPSAISIE) InitElimineBlancs (TPSAISIE)

InitModeEnroule

Déclaration

```
procedure InitModeEnroule(var ESE : EsEnreg;
                          ModeEnroule : TypeModeEnroule);
```

Objet

Sélectionne le mode enroulement pour un écran de saisie.

Commentaires

Cette procédure vous permet de contrôler le fonctionnement du programme de gestion d'écran au cas où le curseur serait positionné sur la dernière ou la première ligne de l'écran de saisie et que l'utilisateur ait activé une commande qui nécessite de le déplacer respectivement sur la précédente ou sur la prochaine ligne . Le mode enroulement par défaut, *ArretAuxBords*, signifie que toute commande de ce type est ignorée et que le curseur reste dans le champ courant. Le meilleur choix est probablement *EnrouleAuxBords* car le curseur sera déplacé de la dernière ligne à la première et vice versa. Une autre possibilité est *SortieAuxBords* : toute instruction située au delà d'un bord sera interprétée comme une commande de sortie. Par exemple si le curseur est sur la ligne supérieure et que l'utilisateur presse ↑, *EcranEdit* sortira avec un résultat de fonction égale à *EsEnrPreced* (édite enregistrement précédent) comme s'il avait pressé PGUP. Si le curseur était sur la derrière ligne et que l'utilisateur ait pressé ↓, le résultat serait *EsEnrSuiv* (édite enregistrement suivant).

Exemple

```
InitModeEnroule(ESE, EnrouleAuxBords);
```

Demande que le curseur soit déplacé sur la première ligne lorsque la dernière ligne est dépassée et vice versa.

Voir aussi

InitAvanceAuto (TPSAISIE)

InitModeMotDePasse

Déclaration

```
procedure InitModeMotDePasse (On : Boolean);
```

Objet

Active ou désactive le mode Mot de Passe.

Commentaires

Cette routine vous permet de désigner des champs particuliers qui seront édtités en mode mot de passe. Dans ce mode d'affichage, le texte entré dans un champ est supprimé et le caractère affecté par l'appel à *InitCarMotDePasse* sera utilisé pour représenter tous les caractères de la chaîne. Cette routine doit être utilisée seulement avec des champs de type string.

Exemple

Dans ce cas, tous les caractères saisis par l'utilisateur seront affichés comme des astérisques.

InitNumerique

Déclaration

```
procedure InitNumerique(On : Boolean);
```

Objet

Active ou désactive l'éditeur numérique.

Commentaires

Cette routine vous permet d'utiliser l'éditeur numérique avec des champs donnés (droite à gauche, ou mode calculette). L'éditeur ne devra être utilisé que pour des champs de type numérique : Byte, Integer, LongInt, Real, Double, etc. Voir "Les trois éditeurs de champs" dans ce chapitre pour plus de détails et SAISIE.PAS pour un exemple d'utilisation de cette procédure.

```
InitNumerique(On);
AjoutChampReal(ESE, 'Salaire horaire', 16, 12, '999,99 F', 16, 25, 12,
               0, 999.99, 0, Fiche.Salaire);
InitNumerique(Off);
```

Dans cet exemple, le champ *Fiche.Salaire* sera modifié en utilisant l'éditeur numérique alors que les champs précédents et suivants seront édités avec l'éditeur chaîne.

Voir aussi

InitAutoNumLock (TPSAISIE)

InitObligatoire

Déclaration

```
procedure InitObligatoire(On : Boolean);
```

Objet

Active ou désactive l'option de saisie obligatoire d'un champ.

Commentaires

Cette routine vous permet de rendre un champ donné obligatoire. Si un contrôle plus poussé est nécessaire, vous devrez utiliser une routine spéciale de contrôle telle que *BlancsInterdit*.

Exemple

```
InitObligatoire(On);
AjoutChampString(ESE, 'Nom', 7, 19, '', 7, 25, 30, 1, nil, Fiche.Nom);
InitObligatoire(Off);
```

Le champ *Fiche.Nom* est obligatoire.

Voir aussi

BlancsInterdit (TPSAISIE)

InitProtection

Déclaration

```
procedure InitProtection(On : Boolean);
```

Objet

Active ou désactive l'option de protection d'un champ.

Commentaires

Cette routine vous permet de protéger un champ donné pour en interdire sa modification. Utilisée pour les champs en lecture seule, cette option est primordiale dans l'élaboration des champs calculés. Pour de nombreux exemples d'utilisation des champs protégés, voir le programme exemple SAISIE.PAS.

Exemple

```
InitProtection(On);
AjoutChampReal(ESE, 'Hebdomadaire', 16, 49, '####.## F', 16, 57, 13,
               0, 0, 0, Fiche.Hebdomadaire);
InitProtection(Off);
```

Dans ce cas, le champ *Fiche.Hebdomadaire* sera protégé car c'est un champ calculé par le programme et non saisi par l'utilisateur.

Voir aussi

ChangeProtection (TPSAISIE) InitPtrApresEdit (TPSAISIE)

InitPtrApresEdit

Déclaration

```
procedure InitPtrApresEdit(var ESE : EsEnreg; P : Pointer);
```

Objet

Pointeur sur la routine appelée après modification d'un champ.

Commentaires

Cette procédure permet d'indiquer la routine appelée lorsqu'un champ a été modifié. Elle sera principalement utilisée avec des champs calculés. Voir "Champs calculés" au début de ce chapitre pour plus de détails et le programme SAISIE.PAS comme exemple d'utilisation de cette procédure.

Exemple

```
{$F+}
procedure GestionChampCalcule(var ESE : ESenreg);
begin
   case ESE.IDCourant of
      {mise à jour de tous les champs affectés par les modifications}
   end;
end;
{$F-}
...
InitPtrApresEdit(ESE, @GestionChampCalcule);
```

Dans cet exemple, la procédure *GestionChampCalcule* appelée lorsqu'un champ a été modifié pourra contrôler *ESE.IDCourant* pour déterminer si d'autres champs sont à modifier.

Voir aussi

InitProtection (TPSAISIE)

InitPtrAvantEdit

Déclaration

```
procedure InitPtrAvantEdit(var ESE : EsEnreg; P : Pointer);
```

Objet

Pointeur sur une routine à appeler juste avant de modifier un champ.

Commentaires

Cette procédure permet de désigner la routine à appeler avant de modifier un champ. L'utilisation la plus courante est l'affichage d'un message d'aide associé au champ à modifier. Voir "Affichage des messages d'aide" au début de ce chapitre pour plus de détails et le programme SAISIE.PAS pour un exemple d'utilisation de cette procédure.

Exemple

```
{$F+}
procedure AfficheMessageAide(var ESE : ESenreg);
begin
   case ESE.IDCourant of
      {affiche le message d'aide pour ce champ}
   end;
end;
{$F-}
...
InitPtrAvantEdit(ESE, @AfficheMessageAide);
```

Dans cet exemple, *AfficheMessageAide* sera appelé avant qu'un nouveau champ soit modifié et utilisera *ESE.IDCourant* pour déterminer le message d'aide à afficher.

InitPtrErreur

Déclaration

```
procedure InitPtrErreur(var ESE : EsEnreg; P : Pointer);
```

Objet

Initialise le pointeur sur la routine à appeler en cas d'erreur.

Commentaires

Cette procédure vous permet de désigner une routine qui sera appelée chaque fois que le gestionnaire d'écrans de saisie rencontrera une erreur. Si aucun gestionnaire d'erreurs n'est installé, l'erreur sera signalée par un bip et aucun message d'erreur ne sera affiché. Voir "Gestionnaire d'erreurs" au début de ce chapitre pour plus de détails et le programme SAISIE.PAS pour un exemple d'utilisation de cette procédure.

Exemple

```
{$F+}
procedure GestionErreur(var ESE : ESenreg; Code : byte; Msg : String);
begin
   {affiche le message d'erreur tant qu'une touche n'est pas pressée}
   {efface la partie de l'écran recouverte par le message}
end;
{$F-}
...
InitPtrErreur(ESE, @GestionErreur);
```

Dans cet exemple, *GestionErreur* sera appelée chaque fois qu'une erreur se produit. Elle affichera *Msg* à l'emplacement désigné sur l'écran, attendra la frappe d'une touche puis effacera la partie de l'écran qui a été recouverte par le message.

InitSortieSurSecondClic

Déclaration

```
procedure InitSortieSurSecondClic(On : Boolean);
```

Objet

Permet un double-clic sur le champ et génère une commande de sortie *EsSortieParClic*.

Commentaires

Cette commande qui est incluse seulement si 'UseMouse' a été défini lorsque TPSAISIE a été compilé, fournit une interface pratique pour les programmes qui utilisent la souris. La meilleure façon d'expliquer ce qu'il faut faire est de donner un exemple concret.

Dans SAISIE.PAS, la touche (F2) est affectée à la commande de sortie définie par l'utilisateur, *EsUtl1*. Si cette commande est activée au moment où *Departement* est le champ courant, SAISIE affiche une liste pick des noms de départements dans laquelle l'utilisateur peut faire son choix. Afin de permettre à la souris de simuler la frappe de (F2), l'appel de fonction qui ajoute le champ *Departement* est encadré par des appels à *InitSortieSurSecondClic*.

```
InitSortieSurSecondClic(On);
AjoutChampString(ESE, 'Département', 10, 18, '##', 10, 25, 02, 4,
                 @VideOuRempli, Departement);
InitSortieSurSecondClic(Off);
```

Cela signifie que *EcranEdit* sortira avec un code retour égal à *EsSortieParClic* si l'utilisateur clique le bouton gauche de la souris (donne la commande *EsSouris*) lorsque 1) *Departement* est déjà le champ courant et 2) Le curseur de la souris est en haut de celui-ci. Cela fournit à SAISIE le moyen d'afficher sa liste pick.

Le nom de cette routine repose sur le fait que le bouton gauche de la souris sera cliqué une deuxième fois dans ce cas : une première fois pour sélectionner le champ courant et une deuxième fois pour générer la commande de sortie.

LibereEcranEdit

Déclaration

```
procedure LibereEcranEdit(var ESE : EsEnreg);
```

Objet

Libère tous les enregistrements de champs associés à *ESE*.

Commentaires

Cette routine libère toute la mémoire allouée dynamiquement lorsque les champs ont été ajoutés sur un écran de saisie et efface toutes les données de *EsEnreg*.

Dans de nombreuses applications, principalement celles qui ne contiennent qu'un seul écran de saisie, cette procédure ne devra jamais être appelée. Elle est par contre indispensable dans les grands programmes comportant plusieurs écrans de saisie, dans lesquels la mémoire dynamique utilisée par l'écran de saisie doit être libérée après chaque session de modification.

Exemple

```
procedure MonEcranSaisie;
var
   ESE : ESenreg;
   Fin : boolean;
   CommandeSortie : TypeES;
begin
   InitEnregES(ESE);
   {ajoût de champs sur l'écran de saisie}
   Fin := False;
   repeat
      CommandeSortie := EcranEdit(ESE, 0, true);
      {traitement de la commande de sortie}
   until Fin;
   LibereEcranEdit(ESE);
end;
```

Utilisation typique de *LibereEcranEdit* : le paramètre de type *EsEnreg* est local à la procédure qui peut être appelée plusieurs fois dans le programme ; la mémoire est allouée au début de la procédure lorsque les champs sont ajoutés. La mémoire est libérée à la fin de la procédure.

Voir aussi

InitEnregEs (TPSAISIE)

MonnaieInternationale

Déclaration

```
Function MonnaieInternationale(CaracForm : Char; MaxChiffre : Byte ;
                               FlottantSiPossible : Boolean;
                               AjouteVirgule : Boolean) : string;
```

Objet

Renvoie à un masque de formatage pour une chaîne monétaire par rapport au paramètre *Country* du DOS.

Commentaire

Cette routine utilise le paramètre *Country* du DOS afin de déterminer non seulement les valeurs appropriées de *CaracDécimal, CaracVirgule, ChnGaucheMonnaie* et *ChnDroitMonnaie* mais aussi définit les masques de formatage appropriés à la monnaie d'un pays en cours. *CaracForm* est le caractère de formatage utilisé (en général 9 ou #). *MaxChiffre* est le nombre de chiffres qui peut apparaître à gauche du point décimal (le DOS spécifie combien de chiffres pourront apparaître sur la droite). *FlottantSiPossible* indique si $ ou C sera utilisé si le symbole monétaire est placé à gauche. *AjouteVirgule* est l'indicateur du séparateur des milliers qui apparaîtront ou non dans le masque de formatage.

Souvenez-vous lorsque vous utiliserez cette routine que la chaîne retournée peut avoir une longueur maximum de 8 + *MaxChiffres* caractères, sans compter les espaces ni les virgules (*Pred(MaxChiffres)Div 3 caractères*). Ceci est dû au fait que les symboles monétaires peuvent avoir une longueur de 5 caractères et qu'il y a généralement 2 chiffres à droite du point décimal. Lors de la conception de vos écrans de saisie de données, n'oubliez pas de laisser suffisamment de place pour le plus grand masque de formatage qu'il sera possible de traiter. Nous vous recommandons d'utiliser cette fonction avec précaution lorsque vos programmes tourneront sous DOS.2.x.

Exemple

```
MonnaieInternationale('9', 6, True, True);
```

Retourne '$999,999.99' pour la monnaie US (*CaracDecimal* = '.', *CaracVirgule* = ',', *ChnGaucheMonnaie* = '$', *ChnDroitMonnaie* = ''), '999,999CCCC' pour la monnaie italienne (*CaracDecimal* inutilisé, *CaracDeuxPoints* = '.', *ChnGaucheMonnaie* = '', *ChnDroitMonnaie* = 'Lit.').

```
MonnaieInternationale('#', 4, False, False);
```

Retourne 'c####.##' pour la monnaie US, 'cccc####.##'.

Voir aussi

DateInternationale (TPDATE) HeureInternationale (TPDATE).

SonneCloche

Déclaration

```
procedure SonneCloche;
```

Objet

Emet un bip sonore.

Commentaires

SonneCloche est appelée si une erreur de contrôle se produit et qu'un gestionnaire d'erreur utilisateur est absent ou si un caractère non valide est saisi. Elle est interfacée de telle manière que les autres parties d'un programme peuvent gérer les erreurs de la même façon que TPSAISIE.

Exemple

```
if ErreurDetectee then begin
   Writeln(MessageErreur);
   SonneCloche;
end;
```

Affiche un message d'erreur et fait sonner la cloche.

Voir aussi

InitBipSurErreur (TPSAISIE) InitCloche (TPSAISIE)

ValideSousChamps

Déclaration

```
procedure ValideSousChamps(MasqueSousChamp : string;
                           var Ec : EnrChamp; var CodeErr : Byte;
                           var ChnErreur : PtrChaine) : Boolean;
```

Objet

Vérifie que les sous-champs d'une chaîne respectent les prescriptions de *MasqueSousChamp*.

Commentaires

Cette routine de bas niveau devrait être appelée seulement par des routines de contrôle de haut niveau. Elle est destinée à assister le programmeur dans l'écriture de ses routines de contrôle personnelles pour les champs contenant plusieurs sous-champs, certains d'entre eux étant obligatoires, d'autres non.

Un *MasqueSousChamp* doit avoir la même longueur que le masque de saisie du champ à contrôler. Les caractères suivants sont significatifs dans *MasqueSousChamp* :

```
r  le sous-champ est obligatoire, les blancs sont interdits
u  le sous-champ est obligatoire sauf si le champ tout entier est vide
p  le sous-champ est optionnel mais ne peut être partiellement rempli.
```

Remarquez que ces caractères ne peuvent être mélangés dans un sous-champ donné. Voir le programme de démonstration SAISIE.PAS pour deux exemples typiques d'utilisation de *ValideSousChamp*.

Exemple

```
const
   MasqueTelephone  : string[14] = '(99-1) 99-99-99-99';
   TelephoneOk : string[14] = '(pp-p) uu-uu-uu-uu';
...
{$F+}
function TelephoneValide(var EC : EnrChamp; var CodeErr : Byte;
                         var ChnErr : PtrChaine) : Boolean;
begin
   TelephoneValide := ValideSousChamps(TelephoneOk,EC,CodeErr,ChnErr);
end;
{$F-}
```

TelephoneValide est une routine de contrôle de haut niveau qui utilise *ValideSousChamp* pour vérifier que les deux derniers sous-champs ne sont pas vides, sauf si le champ tout entier est vide et que le premier sous-champ est plein ou vide.

Voir aussi

BlancsInterdit (TPSAISIE) VideOuRempli (TPSAISIE)

VideOuRempli

Déclaration

```
procedure VideOuRempli(var Ec: EnrChamp;
                       var CodErr : Byte;
                       var ChnErreur : PtrChaine) : Boolean;
```

Objet

Vérifie que tous les sous-champs d'une chaîne sont vides ou remplis.

Commentaires

Cette routine de contrôle vérifie qu'un champ est vide ou rempli. Comme toutes les routines de contrôle de haut niveau, elle ne devra jamais être appelée répétitivement. Elle devra être référencée seulement par son adresse.

Exemple

```
AjoutChampString(ESE, 'Département', 10, 8, '##', 10, 25, 2, 4,
                 @ValidateNoPartial, Fiche.Departement);
```

En spécifiant *VideOuRempli* comme routine de contrôle pour ce champ, nous garantissons que l'utilisateur ne pourra pas saisir une seule lettre pour l'état et nous lui permettons de laisser le champ blanc si l'état est inconnu.

TPDATE : Variables de type Date et de type Heure

Cette unité offre des routines qui permettent d'enregistrer des dates et des heures, de les convertir et de les manipuler dans des calculs.

L'un des points forts de TPDATE est la conversion des dates julienne et des heures en chaînes de caractères et réciproquement. Pour offrir le maximum de souplesse et pour garantir leur portabilité, TPDATE permet de spécifier un masque de formatage pour une chaîne donnée. Par exemple, si la date julienne contient le 2 Janvier 1900 et que vous vouliez la convertir en une chaîne (en utilisant la routine *DateEnChaineDate*), plusieurs masques de formatage sont possibles :

Format	Résultat
'mm/jj/aa'	'01/02/00'
'MM/jj/aa'	' 1/02/00'
'mm/jj/aaaa'	'01/02/1900'
'mm-jj-aaaa'	'01-02-1900'
'jj/mm/aaaa'	'02/01/1900'
'JJ/mm/aaaa'	' 2/01/1900'

Les chaînes d'heures sont traitées de la même façon, mais des caractères de formatage spéciaux sont disponibles avec des formats de 12 heures aussi bien qu'avec des formats militaires de 24 heures. Supposons que vous vouliez convertir une variable heure contenant 1:02 am en une chaîne (avec *HeureEnChaineHeure*) ; plusieurs masques de formatage sont possibles :

Format	Résultat
'hh:mm:ss'	'01:02:00'
'hh:mm'	'01:02'
'hh:mm te'	'01:02 am'
'hh:mmt'	01:02a'

Remarquez que les chaînes d'heures sont toujours traitées en format militaire sauf si un petit 't' apparaît dans le masque de formatage. Un 'e' (em) dans le masque doit toujours être accompagné par un 't' correspondant (heure).

Regardons maintenant quels sont les caractères significatifs dans un masque de formatage :

Caractère	Explications
'm', 'M'	mois (champs date seulement)
'j', 'J'	jour (date)
'a'	année (date)
'/'	'/' ou *CaracSlash* (date, semi-littéral)
'h', 'H'	heures (champs heure seulement)
'm', 'M'	minutes (heure)
's', 'S'	secondes (heure)
':'	':' *CaracDeuxPoints* (date, semi-littéral)
't'	'p', 'P', 'a', 'A' (heure, dépend de HeureMajuscule)
'e'	'm' or 'M' (heure, dépend de HeureMajuscule)

Avec les caractères minuscules, les nombres seront complétés par des zéros et avec des caractères majuscules, ils seront complétés par des espaces. Par exemple, si la date est 1/01/1900, un masque de formatage égal à 'MM/jj/aaaa' produira ' 1/01/1900' alors qu'un masque égal à 'mm/jj/aaaa' produira '01/01/1900'. 'A' n'est pas autorisé comme caractère de formatage car 1) il ne devrait jamais être utilisé, et 2) il entraînerait un conflit avec l'un des caractères de formatage de TPSAISIE qui dépend de TPDATE.

Remarquez que deux de ces caractères '/' et ':', sont appelés caractères "semi-littéraux". Ce qui signifie que lorsque TPDATE rencontre l'un de ces deux caractères dans un masque de formatage, elle les remplace automatiquement avec la valeur courante de *CaracSlash* pour '/' ou *CaracDeuxPoints* (pour ':'). Ce qui vous permet d'écrire un programme contenant plusieurs masques de formatage tels que 'mm/jj/aa' et générant des chaînes de format 'mm-jj-aa' simplement en exécutant l'instruction "*CaracSlash* := '-';". Etant donné un masque de formatage correct et un paramétrage correct pour *CaracDeuxPoints* et *CaracSlash*, TPDATE pourra utiliser la plupart des conventions internationales pour afficher des dates et des heures (Voir les explications des routines *DateInternationale* et *HeureInternationale* ainsi que "Conventions des Monnaies Internationales" plus loin dans le chapitre).

Tous les caractères qui n'apparaissent pas dans la liste ci-dessus sont considérés comme des littéraux lorsqu'ils apparaissent dans un masque de formatage.

TPDATE possède de nombreuses autres possibilités remarquables, telles que des routines de calcul arithmétique sur les dates et les heures, que vous étudierez plus loin dans ce chapitre.

Plage de valeurs pour les dates

TPDATE permet d'enregistrer des dates comme des entiers longs (dans l'intervalle 1/1/1600-12/31/3999) ou des mots (dans l'intervalle 1/1/1900-12/31/2078). La directive de compilation conditionnelle *FourByteDates* définit le mode d'enregistrement. Par défaut, *FourByteDates* est définie et les entiers longs sont utilisés pour enregistrer les dates. Si vous préférez que les dates soient enregistrées comme des mots, pour économiser de la place tout ce que vous devrez faire est de supprimer l'instruction '*{$DEFINE FourByteDates}*'dans TPDEFINE.INC et de recompiler TPDATE. Entre autres choses, cela modifiera les valeurs de plusieurs constantes importantes interfacées par l'unité et décrites plus loin dans ce chapitre. *AnneeMin, AnneeMax, DateMin, DateMax, Date1900* et *DateNonValide*. Dans ce qui suit, nous supposons que *FourByteDates* est définie.

Définition du terme "Date Julienne"

Le terme "Date Julienne" est interprété différemment selon les personnes. Pour les astronomes et les historiens ce terme a une signification technique précise et l'utilisation que nous en faisons fera probablement sourciller certaines personnes car TPDATE ne travaille pas avec des dates juliennes "réelles". Pour les programmeurs, le terme a acquis une signification beaucoup plus large, et correspond à "une date enregistrée comme une valeur entière représentant le nombre de jours écoulés à partir d'un point donné dans le temps". C'est avec cette acceptation (largement admise) que nous utiliserons ce terme.

TPDATE : Déclarations

Types

```
{$IfDef FourByteDates}
Date = LongInt ;
{$Else}
Date = Word ;
{$Endif}
```

Dans TPDATE, les dates sont enregistrées en format Julien et représentent le nombre de jours écoulés depuis le 1er janvier 1600. Toute date inférieure au 31 Décembre 3999, peut être enregistrée sur 4 octets. Cette méthode facilite les calculs de dates puisque le nombre de jours écoulés entre deux dates peut être déterminé facilement par soustraction. Notez cependant, que si la directive de compilation conditionnelle *FourByteDates* n'est pas définie, les dates seront enregistrées comme des mots (deux octets), dans l'intervalle 1/1/1900 à 12/31/2078.

```
ChaineDate = String[LongDate];
```

Représentation d'une date sous forme de chaîne.

```
EnrDateHeure =
      record
         D : Date;
         H : Heure;
      end;
```

Ce type d'enregistrement combine simplement les deux types de dates de base définis par TPDATE, *Date* et *Heure*.

```
TypeJour =(Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi);
```

Type énuméré utilisé pour représenter les jours de la semaine.

```
Heure = LongInt;
```

TPDATE traite les heures comme les dates, et représente une heure donnée par le nombre de secondes écoulées depuis minuit. L'intervalle de variation pour une heure est donc 0 (12 am) à 86399 (11.59.59 pm) puisque le nombre de secondes dans un jour est de 86400 (60*60*24). Cette méthode permet de gagner de la place et facilite les calculs arithmétiques sur les heures.

Constantes

```
{$IfDef FourByteDates}
DateNonValide = $FFFFFFFF;
{$Else}
DateNonValide = $FFFF;
{$Endif}
```

Cette valeur est utilisée pour représenter une date non valide telle que 1/32/1988.

```
HeureNonValide = $FFFFFFFF;
```

Cette valeur est utilisée pour représenter une heure du jour non valide telle que 12:61/00.

```
CaracDeuxPoints : Char = ':' ;
```

Cette constante typée définie quel caractère sera utilisé comme séparateur dans une chaîne heure. Par exemple, si *HeureEnChaineHeure* est passée sous forme d'un masque de formatage égal à 'hh:mm:ss' et que *CaracDeuxPoints* vaut '.' la chaîne sera retournée avec le format 'hh.mm.ss'.

```
{$IfDef FourByteDates}
Date1900 = $0001AC05;
{$Else}
Date1900 = $000;
{$Endif}
```

Cette constante contient la date Julienne égale à 1/1/1900.

```
LongDate = 20;
```

Longueur maximum de la chaîne de format passée à une routine dans TPDATE.

```
ChaineJour : array[TypeJour] of string[9] =
('Dimanche','Lundi','Mardi','Mercredi','Jeudi','Vendredi','Samedi');
```

Ce tableau peut être utilisé pour convertir des variables de type *TypeJour* en chaîne de caractères. Par exemple, si *JourSemaine* = Samedi, *S* prendra la valeur 'Samedi', si l'instruction '*S : = ChaineJour[JourSemaine]*' est exécutée.

Vous pouvez manipuler les noms de jours sous forme abrégée, en utilisant l'instruction 'Copy(S,1,3)'

```
HeureSeule            = 'H';
MinuteSeule           = 'M';
SecondeSeule          = 'S';
HeureSeuleMajusc      = 'H';
MinuteSeuleMajusc     = 'M';
SecondeSeuleMajusc    = 'S';
TempsSeulement        = 'T';
EmSeulement           = 'E';
SeparateurDeuxPoints  = ':';
```

Ces caractères de formatage sont les seuls autorisés dans un masque de formatage d'une chaîne heure. Les caractères de formatage majuscules entraîneront une complémentation à gauche avec des espaces (' 1:11:00'), alors que les caractères de formatage minuscules entraîneront une complémentation à gauche avec zéro ('01:11:00'). Voir le début de ce chapitre pour plus de détails.

```
{$IfDef FourByteDates}
DateMin = $00000000;        {= 01/01/1600}
DateMax = $000D6025;        {= 12/31/3999}
{$Else}
DateMin = $0000;            {= 01/01/1900}
DateMax = $FF62;            {= 12/31/2078}
{EndIf}
```

Ces constantes définissent l'intervalle de variation des dates pour des variables de type Date.

```
HeureMin = 0;   {00:00:00 am}
HeureMax = 86399;   {23:59:59 pm}
```

Ces constantes définissent l'intervalle de variation des heures pour des variables de type Heure.

```
{$IfDef FourByteDates}
AnneeMin = 1600;
AnneeMax = 3999;
{$Else}
AnneeMin = 1900;
AnneeMax = 2078;
{$EndIf}
```

Ces constantes définissent l'intervalle de variation des années pour les variables dates.

```
MoisSeul          = 'm';
JourSeul          = 'j';
AnSeul            = 'a';
MoisSeulMajusc    = 'M';
JourSeulMajusc    = 'J';
SeparateurSlash   = '/';
```

Caractères de formatage valides dans des masques de formatage pour des chaînes dates. Les caractères de formatage majuscules entraînent la complémentation à gauche avec des espaces (' 1/11/1900'), alors que les caractères de formatage minuscules entraînent une complémentation à gauche avec '0' ('01/11/1900'). Voir le début de ce chapitre pour plus de détails.

```
ChaineMois : array[1..12] of string[9] =
('Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet',
'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre');
```

Ce tableau peut être utilisé pour convertir des variables représentant des mois en chaîne de caractères. Par exemple, si *Mois = 1, S* prendra la valeur 'Janvier' si l'instruction '*S := ChaineMois[Mois]*' est exécutée. Les noms de mois peuvent être abrégés en utilisant l'instruction '*Copy(S,1,4)*'

```
SecondesDansJour = 86400;
```

Nombre de secondes dans un jour.

```
CaracSlash : Char = '/';
```

Ces constantes typées déterminent les caractères utilisés comme séparateurs dans des chaînes dates. Par exemple, si *DateEnChaineDate* est passé comme un masque de formatage égal à 'mm/jj/aa' et que *CaracSlash* vaut '-', la chaîne sera retournée avec le format 'mm-jj-aa' .

```
HeureMajuscule : Boolean = False;
      {si vrai, 't' et 'e' forceront des majuscules}
```

Cette valeur modifie le fonctionnement des routines qui renvoient des représentations de chaînes d'heures. Si cette variable vaut "True", et que le masque de formatage passé à la routine contient 't' et/ou 'e', la chaîne renvoyée contiendra 'P', 'PM', 'A', ou 'AM' plutôt que leurs représentations minuscules équivalentes.

Aujourdhui

Déclaration

```
function Aujourdhui : Date;
```

Objet

Renvoie la date du jour sous forme d'une date Julienne.

Commentaires

Cette fonction appelle le DOS pour connaître la date du jour, puis convertit les valeurs retournées (jour, mois, an), en une variable Date.

Exemple

```
Writeln('Nous sommes un ', ChaineJour[JourSemaine(Aujourdhui)]);
```

Affiche le jour de la semaine.

Voir aussi

JourSemaine (TPDATE) AujourdhuiEnChaine (TPDATE).

AujourdhuiEnChaine

Déclaration

```
function AujourdhuiEnChaine(Format : ChaineDate) : ChaineDate;
```

Objet

Renvoie la date du jour sous forme d'une chaîne avec le format spécifié.

Commentaires

Cette fonction appelle simplement *Aujourdhui* pour obtenir la date courante, et la convertit en une chaîne avec le format spécifié.

Exemple

```
Writeln('Nous sommes ', AujourdhuiEnChaine('jj/mm/aaaa'));
```

Affiche la date du jour.

Voir aussi

Aujourdhui (TPDATE).

ChaineDateEnDate

Déclaration

```
function ChaineDateEnDate(Format, C : ChaineDate) : Date ;
```

Objet

Convertit *C* de type *ChaineDate* en date julienne.

Commentaires

Etant donnée une date représentée sous forme d'une chaîne de caractères (C), cette routine convertit cette date en date julienne. *C* doit être une chaîne de même longueur et de même structure que format. Si la date représentée par *C* est non valide, *DateNonValide* est retournée.

Exemple

```
MaChaineDate := '01/01/1900';
MaDate := ChaineDateEnDate('jj/mm/aaaa', MaChaineDate);
```

Dans ce cas, *MaDate* sera égale à la constante prédéfinie Date1900.

Voir aussi

ChaineDateEnJMA (TPDATE) DateEnChaineDate (TPDATE).

ChaineDateEnJMA

Déclaration

```
function ChaineDateEnJMA(Format, C : ChaineDate;
                         var J,M,A : Integer) : Boolean;
```

Objet

Extrait les jours, mois et années de *C* et renvoie "True" si la chaîne est valide.

Commentaires

Cette routine extrait les jour, mois et année de *C* de type *ChaineDate*, et renvoie "True" si la date est valide. *C* doit être une chaîne de même longueur et de même structure que *Format*.

Exemple

```
MaChaineDate := '01/01/1900';
DateOk := ChaineDateEnJMA('jj/mm/aaaa', MaChaineDate, J, M, A);
```

Dans ce cas, *J* vaudra 1, *M* 1, *A* 1900 et *DateOk* vaudra "True".

Voir aussi

ChaineDateEnDate (TPDATE) JMAEnChaineDate (TPDATE)
DateEnChaineDate (TPDATE)

ChaineHeureCourante

Déclaration

```
function ChaineHeureCourante(Format : ChaineDate) : ChaineDate;
```

Objet

Renvoie l'heure courante sous forme d'une chaîne de caractères avec le format spécifié.

Commentaires

Cette fonction appelle simplement *HeureCourante*, pour obtenir l'heure du jour puis la convertit en une chaîne avec le format spécifié.

Exemple

```
Writeln('Il est ', ChaineHeureCourante('hh:mm:ss te'));
```

Affiche l'heure courante.

ChaineHeureEnHms

Déclaration

```
function ChaineHeureEnHms(Format, C : ChaineDate) : Heure;
```

Objet

Convertit *C* en une variable heure.

Commentaires

Etant donnée une chaîne heure (C), cette routine la convertit en une variable heure. *C* doit être une chaîne de même longueur et de même structure que Format. Si l'heure représentée par *C* est non valide, *HeureNonValide* est retournée.

Exemple

```
MaVarHeure := ChaineHeureEnHMS('hh:mm:ss', '01:02:03');
```

La variable *MaVarHeure* vaudra 1:02:03 am.

Voir aussi

HeureEnChaineAmPm (TPDATE) HeureEnChaineHeure (TPDATE).

ChainePourTriEnDate

Déclaration

```
function ChainePourTriEnDate(C : String) : Date;
```

Objet

Convertit une chaîne utilisable dans un tri en une date.

Commentaires

Etant donnée une chaîne utilisable dans un tri (C), produite par *DateEnChainePourTri*, cette routine la convertit en date Julienne.

Exemple

```
var
   MaChainePourTri : string[4];
   MaDate          : Date;
...
MaChainePourTri := DateEnChainePourTri(MaDate);
...
MaDate := ChainePourTriEnDate(MaChainePourTri);
```

Convertit *MaDate* en chaîne utilisable dans un tri et réciproquement.

Voir aussi

DateEnChainePourTri(TPDATE)

ChainePourTriEnHeure

Déclaration

```
function ChainePourTriEnHeure(C : String) : Heure;
```

Objet

Convertit une chaîne utilisable dans un tri en une variable heure.

Commentaires

Etant donnée une date utilisable dans un tri (C), produite par *HeureEnChainePourTri*, cette routine la convertit en une variable heure.

Exemple

```
var
   MaChainePourTri : string[4];
   MonHeure        : Heure;
...
MaChainePourTri := HeureEnChainePourTri(MonHeure);
...
MonHeure := ChainePourTriEnHeure(MaChainePourTri);
```

Convertit *MonHeure* en une chaîne utilisable dans le tri et réciproquement.

Voir aussi

HeureEnChainePourTri (TPDATE).

DateEnChaineDate

Déclaration

```
function DateEnChaineDate(Format : ChaineDate ;
                          DateJulienne : Date) :ChaineDate;
```

Objet

Convertit une date julienne en une chaîne de format spécifié par *Format*.

Commentaires

Etant donnée une date Julienne, cette routine la convertit en une chaîne de format spécifié par *Format*. Remarquez que si *DateJulienne* = *DateNonValide*, les mois, jours et années seront représentés par des espaces.

Exemple

```
MaChaineDate := DateEnChaineDate('jj/mm/aaaa', MaVarDate);
```

Convertit la date en une chaîne de format 'jj/mm/aa'

DateEnChainePourTri

Déclaration

```
function DateEnChainePourTri(DateJulienne : Date) : string;
```

Objet

Convertit une date en une chaîne que l'on peut trier.

Commentaires

Etant donnée une date Julienne, cette routine la convertit en une chaîne qui peut être utilisée par des routines de tris et d'indexation comme celles de *Turbo Database Toolbox*. La chaîne produite a une longueur de 4 octets et n'est pas imprimable.

Exemple

```
var
   MaChainePourTri : string[4];
...
MaChainePourTri := DateEnChainePourTri(MaDate);
```

Convertit *MaDate* en chaîne que l'on peut trier.

Voir aussi

ChainePourTriEnDate (TPDATE)

DateEnJMA

Déclaration

```
procedure DateEnJMA(DateJulienne: Date; var Jour, Mois, An: Integer);
```

Objet

Convertit une date julienne en jour, mois, année.

Commentaires

Etant donnée une date Julienne, cette routine la renvoie en nombre de jours, mois et années. Si *DateJulienne* = *DateNonValide*, *Jour*, *Mois* et *An* vaudront 0.

Exemple

```
var
   Jour, Mois, An : Integer;
...
DateEnJMA(Aujourdhui, Jour, Mois, An);
```

Initialise *Jour*, *Mois* et *An* à la date du jour.

Voir aussi

JMAEnDate(TPDATE).

DateInternationale

Déclaration

```
function DateInternationale(AnneeComplete, CompleteAvecZero : Boolean)
                                : ChaineDate;
```

Objet

Renvoie un masque de formatage pour une chaîne date en fonction du paramètre *Country* du DOS.

Commentaires

Cette routine utilise le paramètre *Country* du DOS, afin de déterminer la valeur appropriée de *CaracSlash*, mais aussi le masque de formatage adapté au pays en cours.

Le paramètre *AnneeComplete* indique si le masque de formatage devra comporter un siècle ('aaaa'), aussi bien qu'une année ('aa'). *CompleteAvecZero* indique si le premier sous-champ dans le masque de formatage devra être complété avec des zéros ou des espaces.

Nous vous recommandons d'utiliser cette fonction avec précaution dans des programmes qui fonctionnent sous DOS 2.x.

Exemple

```
DateInternationale(True, True);
```

Retourne 'mm/jj/aaaa' pour une date US (*CaracSlash* = '-'), 'aaaa/mm/jj' pour une date suédoise (*CaracSlash* = '-').

```
DateInternationale(False, False);
```

Retourne 'MM/jj/aa' pour une date US et 'JJ/mm/aa' pour une date suisse (*CaracSlash* = '.').

Voir aussi

MonnaieInternationale(TPSAISIE) HeureInternationale(TPDATE)

DiffDate

Déclaration

```
procedure DiffDate(Date1, Date2 : Date;
                   var Jours, Mois, Ans : Integer);
```

Objet

Renvoie la différence entre deux dates juliennes dans les paramètres *Jours*, *Mois* et *Ans*.

Commentaires

Etant données deux dates julienne valides *Date1* et *Date2*, cette procédure calcule la différence absolue entre ces deux dates et renvoie le résultat dans *Jours*, *Mois* et *Ans*. L'ordre de *Date1* et *Date2* est sans importance.

Exemple

```
var
   J, M, A : Integer;
...
DiffDate(AujourdHui, AujourdHui+1, J, M, A);
```

Dans ce cas, *Jours* vaudra 1, alors que *Mois* et *Annees* vaudront 0.

Voir aussi

DiffDateEnJourSec(TPDATE) DiffHms(TPDATE)

DiffDateEnJourSec

Déclaration

```
procedure DiffDateEnJourSec(Dt1, Dt2 : EnrDateHeure;
                            var Jours : Word;
                            var Secondes : LongInt);
```

Objet

Renvoie la différence en jours et secondes entre deux dates.

Commentaires

Etant données deux variables de type *EnrDateHeure*, *Dt1* et *Dt2*, cette routine renvoie la différence absolue entre ces deux dates en jours et secondes. L'ordre de *Dt1* et *Dt2* est sans importance.

Exemple

```
var
   DH1, DH2: EnrDateHeure;
   Jours : Word;
   Secs : LongInt;
...
DH1.D := Aujourdhui;      DH1.H := HeureCourante;
DH2.D := DH1.D + 1;  DH2.H := DH1.H + 1;
DiffDateEnJourSec(DH1, DH2, Jours, Secs);
```

Dans ce cas, *Jours* et *Secondes* vaudront 1.

```
DH2.H := DH1.H - 1;
DiffDateEnJourSec(DH1, DH2, Jours, Secs);
```

Jours vaudra maintenant 0 et Secondes 86399 (SecondesDansJour-1).

Voir aussi

DiffDate(TPDATE) DiffHms(TPDATE)

DiffHms

Déclaration

```
procedure DiffHms(Heure1, Heure2 : Heure;
                  var Heures, Minutes, Secondes : Byte);
```

Objet

Renvoie la différence en heures, minutes et secondes entre deux heures.

Commentaires

Etant données deux variables, *Heure1* et *Heure2* valides, *DiffHms* calcule la différence absolue entre elles, en heures, minutes et secondes. L'ordre de heure1 et heure2 est sans importance.

Exemple

```
var
   Heure1, Heure2 : Heure;
   Heures, Minutes, Secondes : byte;
...
Heure1 := HeureCourante;
Heure2 := IncHMS(Heure1, 1, 2, 3);
DiffHMS(Heure1, Heure2, Heures, Minutes, Secondes);
```

Dans ce cas, *Heures* vaudra 1, *Minutes* 2 et *Secondes* 3.

Voir aussi

DiffDate(TPDATE) DiffDateEnJourSec(TPDATE)

HeureCourante

Déclaration

```
function HeureCourante : Heure;
```

Objet

Renvoie l'heure courante en secondes depuis minuit.

Commentaire

Cette fonction appelle le DOS pour déterminer l'heure courante du jour, puis la convertit en une variable de type *Heure* (heures, minutes, secondes), qui est retournée comme résultat de la fonction.

Exemple

```
var
   Debut,
   Ecoule : Heure;
...
Debut := HeureCourante;
...
Ecoule := HeureCourante - Debut;
if Ecoule < 0 then
   Inc(Ecoule, SecondesDansJour);
```

HeureCourante est utilisée pour calculer le nombre de secondes écoulées entre deux événements.

HeureEnChaineAmPm

Déclaration

```
function HeureEnChaineAmPm(Format : ChaineDate;
                           H : Heure): ChaineDate;
```

Objet

Convertit H en une chaîne avec le format spécifié. Heure sera toujours affichée avec le format am/pm.

Commentaires

Cette routine est semblable à *HeureEnChaineHeure* mais représente *H* avec le format am/pm, même si le masque de formatage ne l'indique pas. Remarquez que si *Heure* = *HeureNonValide*, les heures, minutes et secondes seront représentées par des espaces.

Exemple

```
MaChaineHeure1 := HeureEnChaineAmPm('hh:mm:ss', 0);
MaChaineHeure2 := HeureEnChaineHeure('hh:mm:ss', 0);
```

MaChaineHeure1 vaudra '12:00:00' et *MaChaineHeure2* vaudra '00:00:00'.

Voir aussi

HeureEnChaineHeure(TPDATE).

HeureEnChaineHeure

Déclaration

```
function HeureEnChaineHeure(Format : ChaineDate;
                            H : Heure) : ChaineDate;
```

Objet

Convertit *H* en une chaîne avec le format spécifié.

Commentaires

Etant donnée une heure du jour (*H*), cette routine la convertit en une chaîne avec le format spécifié. Remarquez que si *H* = *HeureNonValide*, les heures, minutes et secondes seront représentées par des espaces.

Exemple

```
MaChaineHeure := HeureEnChaineHeure('hh:mm:ss', MaVarHeure);
```

Convertit la variable *MaVarHeure* en une chaîne de la forme 'hh:mm:ss'.

HeureEnChainePourTri

Déclaration

```
function HeureEnChainePourTri(H : Heure) : String;
```

Objet

Convertit une variable heure en une chaîne utilisable dans un tri.

Commentaires

Etant donnée une variable heure (*H*), cette routine la convertit en une chaîne qui peut être utilisée par des routines de tri et d'indexation comme celles de *Turbo Database Toolbox*. La chaîne produite a une longueur de 4 octets et n'est pas imprimable.

Exemple

```
var
   MaChainePourTri : string[4];
   HeureDebut      : Heure;
...
MaChainePourTri := HeureEnChainePourTri(HeureDebut);
```

Convertit *HeureDebut* en une chaîne utilisable dans le tri.

Voir aussi

ChainePourTriEnHeure (TPDATE).

HeureEnHms

Déclaration

```
procedure HeureEnHms(H : Heure; var Heures, Minutes, Secondes : Byte);
```

Objet

Convertit une variable de type *Heure* en heures, minutes, secondes.

Commentaires

Etant donnée une variable heure (*H*), cette routine extrait les heures, minutes et secondes. Si *Heure* = *HeureNonValide*, *Heure*, *Minutes* et *Secondes* vaudront 0.

Exemple

```
var
   Heures, Minutes, Secondes : byte;
...
HeureEnHMS(HeureCourante, Heures, Minutes, Secondes);
```

Initialise *Heures*, *Minutes* et *Secondes* avec l'heure courante.

Voir aussi

HmsEnSecondes (TPDATE).

HeureInternationale

Déclaration

```
function HeureInternationale(AvecSecondes, CompleteAvecZero : Boolean;
                             EspaceSupl, AvecEm : Boolean) : ChaineDate;
```

Objet

Renvoie un masque de formatage pour une chaîne heure, en fonction du paramètre *Country* du DOS.

Commentaires

Cette routine utilise le paramètre *Country* du DOS pour déterminer la valeur appropriée de *CaracDeuxPoints* mais aussi le masque de formatage adapté au pays en cours.

AvecSecondes indique si le masque comportera les secondes aussi bien que les heures et les minutes. *CompleteAvecZero* indique si les heures du sous-champ devront être complétées avec des zéros ou des espaces. *AvecEm* indique si 'te' devra être ajouté ou non à la fin de la chaîne. Et si 'te' est ajouté, *EspaceSupl* indique si un espace devra être placé devant celui-ci ou non.

Contrairement aux versions du DOS supérieur à 3.0, les versions DOS 2.x ne fournissent aucune information sur la façon dont les chaînes heures doivent être formatées. Nous vous recommandons de ne pas utiliser cette fonction dans des programmes qui doivent fonctionner sous DOS.2.

Exemple

```
HeureInternationale(True, True, True, True);
```

Retourne 'HH:mm:ss te' pour les USA (*CaracDeuxPoints* = ':') et 'HH.mm.ss' pour la Norvège (*CaracDeuxPoints* = '.').

```
HeureInternationale(False, False, False, False);
```

Retourne 'hh:mm' pour les USA (*CaracDeuxPoints* = ':') et 'hh.mm' pour le Danemark (*CaracDeuxPoints* = '.').

Voir aussi

MonnaieInternationale (TPSAISIE) DateInternationale (TPDATE)

HmsEnSecondes

Déclaration

```
function HmsEnSecondes(Heure, Minute, Seconde : Byte) : Heure;
```

Objet

Convertit *Heure, Minute* et *Seconde* en une variable Heure.

Commentaires

Etant donnée une heure du jour en *Heure, Minute* et *Seconde,* cette routine renvoie l'heure écoulée en secondes depuis minuit. *Heure* doit être en format militaire (0-23). Si l'heure est non valide, *HeureNonValide* est retournée.

Exemple

```
VarHeure := HMSenSecondes(1, 2, 3);
```

VarHeure sera égale à 3723 soit (1*60*60)+(2*60)+3 ou encore 01:02:03.

Voir aussi

HeureEnHms (TPDATE)

IncDate

Déclaration

```
function IncDate(DateJulienne : Date; Jour, Mois, An : Integer): Date;
```

Objet

Ajoute (ou soustrait) le nombre de jours, de mois et d'années spécifiés à une date.

Commentaires

Etant donnée une date julienne valide, cette routine la recalcule en fonction du nombre de jours, de mois et d'années spécifiés. *Mois* et *An* sont ajoutés avant les jours. Aucun contrôle de débordement n'est effectué puisque le résultat est une date dans l'intervalle *DateMin..DateMax.*

Exemple

```
UnAnApres := IncDate(AujourdHui, 0, 0, 1);
```

UnAnApres est la date julienne égale à la date du jour plus 1 année.

```
UnAnAvant := IncDate(AujourdHui, 0, 0, -1);
```

UnAnAvant est la date julienne égale à la date du jour moins 1 année.

IncHms

Déclaration

```
function IncHms (H : Heure; Heures, Minutes, Secondes : Byte): Heure;
```

Objet

Ajoute le nombre d'heures, minutes, et secondes spécifiés à *H*.

Commentaires

Etant donnée l'heure du jour (*H*), cette routine ajoute le nombre d'heures, minutes et secondes spécifiés à cette heure. Le résultat est recalculé éventuellement à partir de minuit.

Exemple

```
Maintenant := HMSenSecondes(22, 0, 0); {10:00:00 pm}
UneHeurePlusTard := IncHMS(Maintenant, 1, 0, 0);
```

UneHeurePlusTard vaudra 11:00:00 pm.

```
TroisHeuresPlusTard := IncHMS(Maintenant, 3, 0, 0);
```

TroisHeuresPlusTard vaudra 1:00:00 am.

JMAEnChaineDate

Déclaration

```
function JMAEnChaineDate(Format : ChaineDate;
                         Jour, Mois, An : Integer ) : ChaineDate;
```

Objet

Formatage de *Jour*, *Mois* et *An*.

Commentaires

Etant donnés *Jour*, *Mois* et *An*, cette routine renvoie la chaîne représentative de la date avec le format spécifié par *Format*. Si *An* est inférieur à 100, cette date est considérée comme commençant en 1900.

Exemple

```
MaChaineDate := JMAenChaineDate('jj/mm/aaaa', 12, 1, 1964);
```

Initialise *MaChaineDate* à '12/01/1964'

Voir aussi

ChaineDateEnJMA(TPDATE)

JMAEnDate

Déclaration

```
function JMAEnDate(Jour, Mois, An : Integer) : Date;
```

Objet

Convertit *Jour, Mois, An* en une date julienne.

Commentaires

Etant donné une date sous la forme *Jour, Mois, An*, cette routine la convertit en une date julienne. Si la date est non valide, *DateNonValide* est renvoyée. Si *An* est inférieur à 100, elle sera considérée comme commençant en 1900.

Exemple

```
DateSouvenir := JMAenDate(12, 1, 1964);
```

DateSouvenir contient maintenant le nombre de jours entre le 1er Janvier 1600 et le 12 janvier 1964.

Voir aussi

DateEnJMA(TPDATE)

JourSemaine

Déclaration

```
function JourSemaine(DateJulienne : Date) : TypeJour;
```

Objet

Renvoie le jour de la semaine pour la date spécifiée.

Commentaires

Etant donnée une date julienne, cette routine renvoie le jour correspondant de la semaine. *DateJulienne* doit être une date valide.

Exemple

```
WriteLn('Nous sommes un ', ChaineJour[JourSemaine(Aujourdhui)]);
```

Affiche le jour de la semaine.

VerifieDate

Déclaration

```
function VerifieDate(Jour, Mois, An : Integer) : Boolean ;
```

Objet

Vérifie si *Jour, Mois,* et *An* correspondent à une date valide.

Commentaires

Cette routine est destinée principalement aux usages internes, mais vous pouvez l'utilisez si vous le souhaitez. Etant donnés *Jour, Mois, An,* elle renvoie "True" si la date est valide et "False" sinon. Si *An* est inférieur à 100, l'année est supposée commencer en 1900. Si An est supérieur ou égal à 100, elle doit appartenir à l'intervalle 1600..39999.

Exemple

```
DateOk := VerifieDate(1, 1, 1598);
```

DateOk vaudra "False" puisque l'année est hors de l'intervalle.

```
DateOk := VerifieDate(4, 31, 1988);
```

DateOk vaudra "False" puisque il y a seulement 30 jours au mois d'avril.

Index

A

AbsBcd 11.4
ActiveAideSouris 2.199
ActiveAttenteEvenement 2.234
ActiveEcranV 2.86
ActiveEditSouris 2.175
ActiveElemMenu 2.125
ActiveFen 2.88
ActiveFenDisj 2.89
ActiveIndexAide 2.201
ActiveMenuSouris 2.126
ActivePickSouris 2.272
ActiveSaisieSouris 3.34
AdapteCouleursFenCompactee 2.16
AddBcd 11.4
AdresseMacro 9.5
AffAide 2.202
AffAidePreced 2.205
AffecteNomMacro 9.6
AffFenCompactee 2.17
AffFenCompacteeEn 2.17
AfficheFen 2.90
AffSouris 2.235
AideEcran 2.206
AjoutBackSlash 4.5
AjoutChampArray 3.35
AjoutChampBcd 3.36
AjoutChampBoolean 3.38
AjoutChampByte 3.39
AjoutChampChar 3.40
AjoutChampChnDate 3.41
AjoutChampChoix 3.42
AjoutChampComp 3.44
AjoutChampDate 3.45
AjoutChampDbl 3.46
AjoutChampEmboite 3.47
AjoutChampExt 3.48
AjoutChampHeure 3.49
AjoutChampInt 3.50
AjoutChampLong 3.50
AjoutChampOuiNon 3.51
AjoutChampReal 3.51
AjoutChampSgl 3.52
AjoutChampShort 3.52
AjoutChampString 3.53
AjoutChampUtl 3.55
AjoutChampWord 3.56
AjoutFenBiblio 2.18
AlloueEcran 2.295
AlloueMacro 9.7
AlloueMemExt 8.21
AllouePagesEms 8.5
AllouePile 6.8
AlloueT 10.5, 10.22, 10.35
AnalyseChemin 5.4
AnuleImpFich 5.5
AnuleImpTousFichs 5.6
AppelAncISR 12.8
AppelFar 12.9
AppelNear 12.10
ApureT 10.37
ArcTanBcd 11.5
ArrangeMots 4.6
AscCaracChn 4.64
AscCentre 4.64
AscCentreCar 4.65
AscComplete 4.66
AscCompleteCar 4.67
AscCompleteGauche 4.68
AscCompleteGaucheCar 4.69
AscDepuisTas 4.70
AscEff 4.71
AscEffDebut 4.72
AscEffFin 4.73
AscEnChn 4.74
AscMajusc 4.75
AscMinusc 4.76
AscSurTas 4.77
AssignCrt 2.296
AssociePageEms 8.6
Aujourdhui 3.103
AujourdhuiEnChaine 3.103

B

BcdEnReel 11.5
BlancsInterdits 3.57
BoutonSourisPresse 2.236
BoutonSourisRelache 2.237

C

CacheSouris 2.238
CadreDroite 2.19
CadreFen 2.20
CaracMacro 9.8
Centre 4.7
CentreCar 4.7
CentreVite 2.21
ChaineCarac 4.8
ChaineDateEnDate 3.104
ChaineDateEnJMA 3.104
ChaineDuTas 4.9
ChaineHeureCourante 3.105
ChaineHeureEnHms 3.105
ChainePourTriEnDate 3.106
ChainePourTriEnHeure 3.107
ChaineSurTas 4.10
ChainInt 6.9
ChangeAttr 2.22
ChangeAttrFen 2.23
ChangeHotKey 6.34
ChangeProtection 3.58
ChargeT 10.6, 10.23, 10.38
ChargeTV 10.7
ChercheAide 2.208
ChercheElemMenu 2.128
ChnBcd 11.6
ChnEnAsc 4.78
ChnEnEntier 4.11
ChnEnLongEntier 4.12
ChnEnMot 4.13
ChnEnReel 4.14
ChnExpBcd 11.7
ChnMajusc 4.15
ChnMinusc 4.15
ChoixMenu 2.129
ChoixPick 2.274
CodesIdentFichsOuverts 5.7
ColsActuelles 2.296
ColsEcranV 2.91
CompacteBiblio 2.24
CompacteFen 2.25
CompareAsc 4.78
CompareAscMajusc 4.79
CompareChaine 4.16
CompareChnMajusc 4.17
CompareEnsLettres 4.18
CompareStruct 4.20
CompBiblioMenu 2.132
Complete 4.21
CompleteCar 4.22
CompleteGauche 4.23
CompleteGaucheCar 4.24
CompteMots 4.25
ConcatAsc 4.80
ConcatChn 4.81
ConstruitTableBM 4.26
ContrlClavier 2.26
ContrlEtatMenu 2.133
CoordXAbs 2.27
CoordXY 2.28
CoordYAbs 2.29
CopieAsc 4.82
CopieEcran 2.297
CopieFen 2.299
CorrigeNomChemin 4.27
CosBcd 11.8
CouleurCouleurs 2.301
CouleursMono 2.302
CreeBiblio 2.30
CreeEcranV 2.92
CreeEnsLettres 4.28
CursBloc 2.31
CursCache 2.31
CursDebFin 2.32
CursNormal 2.33
CursPlat 2.33
CursSourisBloc 2.238
CursSourisCache 2.239
CursSourisHard 2.240
CursSourisNormal 2.241
CursSourisPlat 2.241
CursSourisSoft 2.242

D

DateEnChaineDate 3.107
DateEnChainePourTri 3.108
DateEnJMA 3.109
DateInternationale 3.110
DeallouéAsc 4.83
DeallouéChaine 4.29
DeallouéCodeIdentEms 8.8
DeallouéEcran 2.303

DeallouеFen 2.93
DeallouеMacro 9.8
DeallouеMemExt 8.22
DeallouеMenu 2.134
DeallouеPile 6.10
DeallouеT 10.8, 10.24, 10.39
DefinitEcran 2.304
DefinitMacro 9.9
DefinitPop 6.35
DefinitProcPop 6.36
DeplaceEcran 2.34
DeplaceEcrVDansFen 2.94
DeplaceFen 2.96
DeplaceFenDansEcrV 2.97
DeplaceFenHoriz 2.305
DeplaceFenVerti 2.306
DeplaceMemExt 8.23
DerouleFen 2.98
DerouleFenBas 2.35
DerouleFenHaut 2.36
DesactiveAideSouris 2.210
DesactiveAttenteEvenement 2.243
DesactiveEcranV 2.99
DesactiveEditSouris 2.175
DesactiveElemMenu 2.135
DesactiveIndexAide 2.211
DesactiveMenuSouris 2.137
DesactivePickSouris 2.276
DesactivePop 6.37
DesactiveResid 6.38
DesactiveSaisieSouris 3.59
DessineChamp 3.59
DessineEcranEdit 3.60
DessineMenuPile 2.138
DiffDate 3.111
DiffDateEnJourSec 3.112
DifferentBcd 11.9
DiffHms 3.113
DivBcd 11.9

E

EchangeMots 12.11
EchangeOctets 12.11
EchangeStructs 12.12
EchPileEtAppel 6.11
EchPileEtAppelNear 6.13
EcranEdit 3.61
EcritAsc 4.84
EcritAttr 2.37
EcritAttrFen 2.37
EcritBiblioMenu 2.139
EcritCentre 2.307
EcritEcranV 2.100
EcritElemT 10.9, 10.25, 10.40
EcritFacile 2.38
EcritFacileFen 2.39
EcritFenCompactee 2.40
EcritFichMacro 9.9
EcritMemVite 2.308
EcritRapide 2.309
EcritSecteursDisque 5.8
EcritVite 2.41
EcritViteEtCoupe 2.101
EcritViteFen 2.42
EcritVitePile 2.102
EditeChaine 3.63
EffaceEcranV 2.103
EffaceEntFen 3.65
EffaceFenSup 2.104
EffaceMacros 9.10
EffaceMenu 2.141
EffaceMenuPile 2.142
EffaceSousMenuActuel 2.143
EffBlancs 4.30
EffEspDeb 4.31
EffEspFin 4.32
EgalBcd 11.10
ElemMenu 2.145
EmsPresent 8.9
EmuleInt 6.14
EnleveMacros 9.10
EnleveModule 6.39
EnregistreCoordsFen 2.43
EnregistreCoordsFen 2.310
EnregistrementMacroOff 9.11
EnregistrementMacroOn 9.11
EnregistreT 10.11, 10.27, 10.42
EnregistreTV 10.12
EnvoiFinInt 6.15
Erreur8087 6.53
ErreurT 10.13, 10.28, 10.43
EtatMenu 2.147
Exceptions8087 6.54
ExecDos 5.10
ExisteSurChemin 5.13
ExpBcd 11.11
ExtraitMot 4.33

F

FenEstActive 2.105
FenPick 2.277
FenSouris 2.244
FenSourisPleinEcran 2.245
FermeAide 2.212
FermeBiblio 2.44
FichExiste 5.14
FinEtatImp 5.15
FixeCouleur 2.45
Fonte8x8Sel 2.46
ForceSuffixe 4.34
Formate 4.35
Formate 11.11
FracBcd 11.12

H

HauteurMenu 2.148
HercPresent 2.47
HeureCourante 3.114
HeureEnChaineAmPm 3.115
HeureEnChaineHeure 3.115
HeureEnChainePourTri 3.116
HeureEnHms 3.117
HeureInternationale 3.118
HmsEnSecondes 3.119
HorlogeMs 5.15

I

IdentEstConsole 5.16
IncDate 3.119
IncHms 3.120
IndicateurExiste 12.13
IndicDosActif 6.39
IndicErrDos 6.40
IndicsClavier 2.48
InferieurBcd 11.12
InferieurEgalBcd 11.13
InitAttrChaine 3.65
InitAttrChamp 3.66
InitAttrCtrl 3.67
InitAttrMess 3.68
InitAutoNumLock 3.69
InitAvanceAuto 3.70
InitBipSurErreur 3.71
InitBordCrt 2.49
InitCadreInactif 2.106
InitCaracComplement 3.72
InitCaracMotDePasse 3.73
InitCaracsCadre 2.50
InitCarFond 2.311
InitClign 2.51
InitCloche 3.73
InitCurseurEnFin 3.74
InitDelimiteurs 3.75
InitDta 5.17
InitEffPremCarac 3.76
InitElimineBlancs 3.77
InitEnregEs 3.78
InitEvenementsSouris 2.246
InitFen 2.107
InitFenDisjSup 2.109
InitFenEnt 3.79
InitFenSup 2.110
InitForceMode 3.80
InitHorlogePop 6.41
InitIndicateur 12.14
InitInsereEtPousse 3.81
InitJustifDroite 3.82
InitMacros 9.12
InitMenu 2.150
InitMenuDelai 2.151
InitModeEnroule 3.83
InitModeHerc 2.52
InitModeMenu 2.152
InitModeMotDePasse 3.84
InitNomElemMenu 2.153
InitNumerique 3.84
InitObligatoire 3.85
InitPageActive 2.312
InitPageSouris 2.247
InitPageVisible 2.53
InitPops 6.42
InitPosAide 2.213
InitProtection 3.86
InitPtrApresEdit 3.87
InitPtrAvantEdit 3.88
InitPtrErreur 3.89
InitRatioMickeyEnPixel 2.248
InitSaut 12.15
InitSortieSurSecondClic 3.90
InitSouris 2.249
InitT 10.15, 10.30, 10.45
InitTailleCurs 2.54
InitToucheSelMenu 2.155
InitVecteur 6.16
InitVerif 5.18
InitVideo 2.313

InsereAsc 4.85
InsereChn 4.86
InstalleModule 6.43
Int24Off 6.59
Int24On 6.59
IntBcd 11.13
InterruptsOff 6.17
InterruptsOn 6.17

J

JMAEnChaineDate 3.120
JMAEnDate 3.121
JourSemaine 3.121

L

LanceImpFich 5.19
LanceMacro 9.13
LanceMacroParTouche 9.14
LanceSousNiveau 2.156
LargeurMenu 2.157
LecteurParDefaut 5.20
LibereEcranEdit 3.91
LibereEcranV 2.112
LibereFenCompactee 2.55
LibereVideo 2.314
LigneDebutCurs 2.55
LigneFinCurs 2.56
LignesActuelles 2.315
LignesEcranV 2.113
ListeAide 2.214
LitAsc 4.87
LitAttr 2.57
LitAttrCurs 2.58
LitAttrFen 2.58
LitBiblioMenu 2.158
LitCaracSousCurs 2.59
LitCaractere 2.176
LitChaine 2.177
LitChn 4.93
LitDta 5.21
LitEcranV 2.114
LitElemT 10.16, 10.31, 10.46
LitEntier 2.178
LitEnvironnement 5.22
LitEtatCurs 2.60
LitEtatImp 5.23
LitFenBiblio 2.61
LitFenCompactee 2.62
LitFichMacro 9.15
LitInfoDisque 5.24
LitLigneVite 2.315
LitLongEntier 2.179
LitModeFich 5.25
LitModeVideo 2.63
LitMot 2.181
LitMotTouche 2.64
LitNomElemMenu 2.159
LitNomFic 2.286
LitPageSouris 2.250
LitReel 2.182
LitSecteursDisque 5.26
LitTotalMickey 2.251
LitToucheOuBouton 2.252
LitVite 2.65
LitViteFen 2.65
LnBcd 11.14
LongAsc 4.88
LongEnChn 4.37
LongEntEnBcd 11.14
LongEntierBin 4.38
LongEntierHex 4.38
LongEntierOctal 4.39

M

MacroMajuscule 4.39
MacroMinuscule 4.39
MacrosOff 9.16
MacrosOn 9.17
MarqueMemExt 8.24
MemExtDisp 8.25
Minuscule 4.41
ModeGraphHerc 2.66
ModeMenu 2.160
ModifBloc 5.28
ModifTouches 9.21
ModulePresent 6.44
MonnaieInternationale 3.92
MotBin 4.42
MotHex 4.43
MotMax 12.16
MotMin 12.16
MotOctal 4.43
MotPoidsFaible 12.17
MotPoidsFort 12.17
MotToucheSouris 2.253
MultBcd 11.15

N

NbCodesIdentEmsActifs 8.10
NbLecteurs 5.28
NbMacro 9.17
NomCheminComplet 4.44
NomCheminSeul 4.45
NomFichSeul 4.45
Normalise 12.18
NouvMenu 2.161

O

OctetBin 4.46
OctetHex 4.46
OctetOctal 4.47
OuiOuNon 2.184
OuvreBiblio 2.67
OuvreFicAide 2.216
OuvreMemAide 2.218
OuvreUniteStd 5.29

P

PagesEmsDisp 8.11
PagesEmsPrises 8.12
ParagraphesAConserver 6.45
PermuteMot 12.18
PlaceChaine 2.68
PlaceElement 7.5
PlaceEnXYAbs 2.69
PlaceTab 4.48
PlaceTouche 2.70
PopupsOff 6.46
PopupsOn 6.47
PosAsc 4.89
PosChn 4.90
PosTexte 5.29
PrendElement 7.6
Presence8087 6.55
PrimitiveAlloueT 10.18, 10.48
PrintPresent 5.30
PtrCadrePageEms 8.13
PtrElemMenu 2.163
PtrEnEtendu 8.26
PtrHex 4.49
PtrModuleParNom 6.48

R

RecadreFen 2.115
RechBMSansMajMin 4.50
Recherche 4.52
RechercheBM 4.53
RechercheIndexMacro 9.18
RechercheSansMajMin 4.55
RechTexte 5.31
ReelEnBcd 11.16
ReelEnChn 4.56
ReinitEcran 2.316
ReinitVideo 2.71
RemplitCaracAttr 2.317
RemplitCaracVideo 2.318
RemplitFen 2.319
RemplitFenPick 2.280
RemplitMot 12.19
RemplitStruct 12.20
RemplitVite 2.72
RemplitViteFen 2.72
RendMemExt 8.27
RepriseErreur 12.5
Restaure8087 6.56
RestaureContexteEms 8.15
RestaureCoordsFen 2.73
RestaureCoordsFen 2.319
RestaureEtatCurs 2.73
RestaureEtatSouris 2.254
RestaureFen 2.74
RestaureTousVecteurs 6.18
RestaureVecteur 6.19
RestaureVecteursClavier 5.40
ResultInt24 6.48; 6.60
RetourInt 6.20
RetourneChn 4.93
RetraceMenu 2.164
RoundBcd 11.17

S

SautAncIsr 12.21
SautLong 12.22
Sauve8087 6.57
SauveContexteEms 8.16
SauveEtatSouris 2.255
SauveFen 2.75
SelectEcran 2.320
SelectElemMenu 2.165
SelectLecteur 5.32
SelFonte8X8 2.76

SinBcd 11.18
SonneCloche 3.93
Soundex 4.57
SourisDansFen 2.256
SourisEnXY 2.257
SourisPressee 2.257
SourisX 2.258
SourisXY 2.259
SourisY 2.260
SousMenu 2.166
SoustraitBcd 11.18
SqrBcd 11.19
SqrtBcd 11.19
StatutEmsOK 8.17
SuffixeParDefaut 4.58
SuffixeSeul 4.59
SuperieurBcd 11.20
SuperieurEgalBcd 11.20
SupprimeAsc 4.91
SupprimeFenBiblio 2.77
SupprimePops 6.49
SupprimeProcPop 6.49
SupTab 4.60

T

TailleBuffSouris 2.261
TailleFichTexte 5.32
TermineEtResteResident 6.50
TesteModeHerc 2.78
TexteRapide 2.78
TexteRapideFen 2.79
TotalMemExt 8.28
TotalPagesEms 8.18
ToucheEnChaine 9.19
Tri 7.7
TruncBcd 11.21

U

UpCase 4.61

V

ValBcd 11.22
ValideSousChamps 3.94
VerifieDate 3.122
VerifOn 5.33
VersionDos 5.33
VersionEms 8.19
VertVite 2.79
VertViteFen 2.80
VideBufTexte 5.34
VideOuRempli 3.95

Z

ZeroIndicateur 12.23

Reproduit par INSTAPRINT S.A.
1-2-3, levée de la Loire – LA RICHE – B.P. 5927 – 37059 TOURS Cedex
Tél. 47 38 16 04

Dépôt légal 1er trimestre 1989